Chapter 8

Mandatory Access Control

With discretionary access control (DAC) policies, authorization to perform operations on an object is controlled by the object's owner or by principals whose authority can be traced back to that owner. The goals of an institution, however, might not align with those of any individual. So rules set by the institution (rather than rules set by individuals) are the more natural basis for authorization. Such rules are known as *mandatory access control* (MAC) policies.[1]

Some MAC policies serve specific kinds of institutions by codifying best practices that long preceded computerization. Examples include:

- *Multi-level security policies*, which supports a need-to-know principle for accessing confidential information. Developed for use by the military, diplomatic, and intelligence communities, multi-level security policies have found application in other settings, as well as for protecting integrity (rather than confidentiality) of information.
- *Commercial security policies*, which embody accounting practices to protect against financial fraud by employees, customers, and/or partners. Such policies enforce separation of duty and prevent incorrectly or incompletely performed business processes.

This chapter discusses these MAC policies. It also discusses two popular general-purpose frameworks for specifying MAC policies: *domain and type enforcement* is reminiscent of a DAC access matrix (page 130); *role-based access control* supports access restrictions that derive from responsibilities an organization assigns to roles. A third framework, credentials-based authorization, is discussed in Chapter 9.

8.1 Multi-level Security

In the days when classified information was stored only on paper documents, confidentiality was enforced by restricting access to those documents and trust-

[1] Some authors instead use the term *non-discretionary access control*.

ing people to keep secret what they read. A person could see a document D only if (i) there was reason to trust that person would not divulge the secrets in D and (ii) the contents of D was considered relevant to that person's job.

These *need-to-know* conditions can be formalized in terms of *labels* that both characterize content and define a level of trust for keeping secrets. A *classification* $\mathcal{L}(D)$ is assigned to each document D, a *clearance* $\mathcal{L}(U)$ is assigned to each person U, and a relation $\mathcal{L}(D) \preceq \mathcal{L}(U)$ is defined to hold if and only if conditions (i) and (ii) above are satisfied:

> **Multi-level Document Confidentiality Policy.** A person U is permitted to see a document D only if $\mathcal{L}(D) \preceq \mathcal{L}(U)$ holds. □

Note that if the set of labels is too small (as often will be the case) then relation $\mathcal{L}(D) \preceq \mathcal{L}(U)$ can only approximate "need-to-know". With a small set of labels but a large set of documents, two documents D_1 and D_2 might exist that must be given the same label even though a person might have a "need-to-know" for the contents of D_1 but not for D_2. Thus, $\mathcal{L}(D_2) \preceq \mathcal{L}(U)$ would hold and, because $\mathcal{L}(D_1) = \mathcal{L}(D_2)$ holds, so would $\mathcal{L}(D_1) \preceq \mathcal{L}(U)$ (even though it shouldn't).

At the U.S. Department of Defense (DoD), a classification L is formalized[2] as a pair $\langle S_L, C_L \rangle$:

- S_L is a *sensitivity*. It categorizes worst-case potential damage to national security from disclosure of the document contents:

sensitivity	worst-case potential damage to national security
TS (Top Secret)	exceptionally grave
S (Secret)	serious
C (Confidential)	some
U (Unclassified)	none

  This definition induces a total ordering $\mathsf{U} < \mathsf{C} < \mathsf{S} < \mathsf{TS}$, and we write $\mathsf{X} \leq \mathsf{Y}$ to denote $\mathsf{X} = \mathsf{Y} \vee \mathsf{X} < \mathsf{Y}$.

- C_L is a set of *compartments*.[3] Each compartment is identified by the name of some specific topic area. Names might be self-explanatory (e.g., chem/bio, crypto, or nuclear) or obscure (e.g., Ultra or Umbra).[4] With an

[2] DoD classifications are actually more complicated than what we formalize here. They include an additional sensitivity level FOUO (For Official Use Only) between C and U. Also, they allow markings to impose caveats on information handling. For example, the ORCON (originator controlled) caveat prevents forwarding information without consent of the owner—a form of digital rights management.

[3] DoD does not associate a set of compartments with the lower-sensitivities C and S. So an individual having a C or S clearance is authorized to see all documents classified at or below that sensitivity. However, to simplify the exposition that follows and obtain a uniform structure, we will associate compartments with C and S.

[4] In DoD, names whose meanings are classified are known as *codewords*. For example, Ultra is a codeword used during World War II to label information the Allies obtained by decrypting intercepts of German communications, and Umbra is a more recent (but also now-retired) codeword for the most-sensitive kinds communications intercepts.

obscure compartment name, people who see the name but do not have a "need to know" are kept in the dark about what the name describes.

Example labels for classifications in this DoD scheme include: $\langle \mathsf{C}, \{\mathsf{chem/bio}\}\rangle$, $\langle \mathsf{TS}, \{\mathsf{Umbra}\}\rangle$, and $\langle \mathsf{S}, \{\mathsf{crypto}, \mathsf{nuclear}\}\rangle$.

Each document D receives its classification $\mathcal{L}(D) = \langle S_D, C_D\rangle$ from a *classification authority*, an agency that reads and analyzes the contents of D. S_D is the assessment of damage that could result from revealing the contents of D; C_D is the set of topic areas D covers.

Every person U who might require access to classified documents is granted a clearance $\mathcal{L}(U) = \langle S_U, C_U\rangle$ after submitting to a background investigation that seeks to identify character flaws or exploitable personal circumstances. This background investigation might range from a short interview (for access to Unclassified and Confidential information) to a polygraph test (for access to Top Secret information). Higher values for S_U are assigned when fewer opportunities exist for adversaries to coerce traitorous acts by U; C_U is the set of topics relevant to U's job.

By defining $\langle S, C\rangle \preceq \langle S', C'\rangle$ to be

$$S \le S' \ \wedge \ C \subseteq C' \tag{8.1}$$

we have that $\mathcal{L}(D) \preceq \mathcal{L}(U)$ is equivalent to $S_D \le S_U$ and $C_D \subseteq C_U$. Condition $\mathcal{L}(D) \preceq \mathcal{L}(U)$ in Multi-level Document Confidentiality Policy now implies that U satisfies the desired need-to-know conditions for seeing D: $S_D \le S_U$ implies that a background investigation of U gave reason to believe that U is sufficiently trustworthy not to leak the contents of D, and $C_D \subseteq C_U$ implies that all of the topics covered in document D are relevant to U's job.

Nothing requires that documents be the *content unit* being assigned classification labels. We might instead elect to have each paragraph or each sentence within a document be considered a separate content unit with its own label. A *multi-level document* comprises multiple content units with potentially different classifications. The above Multi-level Document Confidentiality Policy still applies, provided label $\mathcal{L}(D)$ on multi-level document D satisfies $\mathcal{L}(d_i) \preceq \mathcal{L}(D)$ for all content units d_i comprising D. Multi-level Document Confidentiality Policy also could be used for controlling access to an individual content unit d_i by considering individual content units each to be a separate document.

8.1.1 Multi-level Confidentiality

Documents in a computer system are stored in files, which can be accessed only by programs. So files correspond to the documents and programs correspond to the users in Mult-level Document Confidentiality Policy, above. But because a program might copy from one file to another, enforcement no longer is simply a matter of blocking certain reads—some writes have to be blocked.

To simplify the account that follows, assume fixed sets of users, files, and programs. We also assume that the label assigned to each file and to each user

does not change:[5]

> **Tranquility Assumption.** Each file F is assigned a fixed classification $\mathcal{L}(F)$ and each user U is assigned a fixed clearance $\mathcal{L}(U)$. □

A file label $\mathcal{L}(F)$ is considered *sound* if $\mathcal{L}(F) = \langle S_F, C_F \rangle$ implies that the contents of F is not more sensitive than S_F and that all topics F spans are included in C_F. Provided labels assigned to files remain sound despite updates to those files, data processed by a program that user U invokes is guaranteed to be data that U is authorized to read, if the following policy is enforced.

> **Multi-level File Confidentiality Policy.** Programs invoked by a user U do not process data derived from any file F where $\mathcal{L}(U) \preceq \mathcal{L}(F)$ holds. □

In order to enforce this policy, let Pgm be a program that U can invoke. We associate with Pgm a fixed label $\mathcal{L}(Pgm) = \langle S_{Pgm}, C_{Pgm} \rangle$, and we define soundness of $\mathcal{L}(Pgm)$ to mean that information read and written during execution of Pgm is not more sensitive than S_{Pgm} and concerns only those topics listed in C_{Pgm}. $\mathcal{L}(Pgm)$ is selected by some classification authority that determines Pgm is sufficiently trustworthy to handle information classified at or below $\mathcal{L}(Pgm)$. That determination might be made by an expert who has been approved by the classification authority or by an automated analysis regime that has been prescribed by the classification authority. But the label cannot simply be $\mathcal{L}(U)$ assigned to the user U who invokes Pgm.

Notice that Multi-level File Confidentiality Policy will be satisfied if (i) $\mathcal{L}(Pgm) \preceq \mathcal{L}(U)$ holds and (ii) $\mathcal{L}(Pgm)$ is sound. Requirement (i) is equivalent to:

> **MLFC Program Invocation.** $\mathcal{L}(Pgm) \preceq \mathcal{L}(U)$ must hold for a program Pgm executing on behalf of a user U. □

Sometimes satisfying this requires creating a new instance of the program but with a reduced label.

For requirement (ii), it seems plausible to assume that $\mathcal{L}(Pgm)$ would be sound if Pgm reads no information that is classified as too sensitive or concerns inappropriate compartments relative to $\mathcal{L}(Pgm)$. So an attempt to read F by Pgm should be allowed to proceed only if $\mathcal{L}(F) \preceq \mathcal{L}(Pgm)$ holds. A reference monitor that intercepts read operations can enforce such a prohibition on *read-up* attempts, and we have:

> **MLFC Read Restriction.**[6] $\mathcal{L}(F) \preceq \mathcal{L}(Pgm)$ must hold for program Pgm to read a file F. □

[5] This assumption of fixed labels is traditionally referred to as the *strong tranquility principle*. We here call it an "assumption" (rather than a "principle") to make clear that it limits the environment. The term *weak tranquility principle* is used in the literature to denote the assumption stipulating that labels may change during execution provided those changes do not allow violations of the security policy.

[6] This is sometimes called the *simple security condition* in the literature.

But this construction assumes that labels on files are sound. A write to F by Pgm changes the contents of F, after which $\mathcal{L}(F)$ might no longer be sound. In particular, if $\mathcal{L}(Pgm)$ is sound then writes to files F where $\mathcal{L}(F) \preceq \mathcal{L}(Pgm)$ holds could make $\mathcal{L}(F)$ unsound by writing information read from a file that has a higher classification than $\mathcal{L}(F)$. The reference monitor preserves soundness for file labels by enforcing a prohibition against *write-down* attempts.

> **MLFC Write Restriction.**[7] $\mathcal{L}(Pgm) \preceq \mathcal{L}(F)$ must to hold for a program Pgm to write into a file F. □

This completes the set of rules a reference monitor must enforce to ensure that Multi-level File Confidentiality Policy remains satisfied.

***Close Look at Computations.** The analysis just given ignores the transformation that Pgm performs. A closer look reveals that implicit in the definition of soundness for $\mathcal{L}(Pgm)$ are (often overlooked) limitations about those transformations. We explore some of those here.

Rather than considering files to be monolithic, we now view each file F as a multi-level object that comprises a collection of content units. Each content unit f has a classification $\mathcal{L}(f) = \langle S_f, C_f \rangle$ and is accessed separately using a single read or write operation. For f read from a file F, soundness of $\mathcal{L}(F)$ implies that $\mathcal{L}(f) \preceq \mathcal{L}(F)$ holds. Given MLFC Read Restriction $\mathcal{L}(F) \preceq \mathcal{L}(Pgm)$, we conclude by transitivity that $\mathcal{L}(f) \preceq \mathcal{L}(Pgm)$ holds for any content unit f that a program Pgm reads. So if $\mathcal{R}_{Pgm} = \{f_1, f_2, ..., f_n\}$ is the set of content units that Pgm reads during some execution, then we have

$$(\forall f \in \mathcal{R}_{Pgm}\colon\ \mathcal{L}(f) \preceq \mathcal{L}(Pgm)). \tag{8.2}$$

For any set $\mathcal{R}$ of information items, define label $\mathcal{L}^{\sqcup}(\mathcal{R})$ to be the least label that is equal to or larger than[8] the classifications assigned to each content unit $f \in \mathcal{R}$:

$$\mathcal{L}^{\sqcup}(\mathcal{R}) \ = \ \langle \max_{f \in \mathcal{R}}(S_f),\ \bigcup_{f \in \mathcal{R}} C_f \rangle$$

From the definitions of $\mathcal{L}^{\sqcup}(\mathcal{R})$ and $\preceq$, we have for any label L:

$$(\forall f \in \mathcal{R}\colon\ \mathcal{L}(f) \preceq L) \ \Rightarrow\ \mathcal{L}^{\sqcup}(\mathcal{R}) \preceq L \tag{8.3}$$

By instantiating (8.3) with $\mathcal{L}(Pgm)$ for L and with $\mathcal{R}_{Pgm}$ for $\mathcal{R}$, we can use (8.2) and infer

$$\mathcal{L}^{\sqcup}(\mathcal{R}_{Pgm}) \preceq \mathcal{L}(Pgm). \tag{8.4}$$

Each value that Pgm writes is some function over the set $\mathcal{R}_{Pgm}$ of information items Pgm has read. Let $G_F(\mathcal{R}_{Pgm})$ be that function for a content

[7] This is sometimes called the **-property* in the literature, although Bell and La Padula's original formulation of it involved restrictions for reads, for writes, and for appends.

[8] $\mathcal{L}^{\sqcup}(\mathcal{R})$ is thus the *least upper bound* (lub) of $\mathcal{R}$.

unit being written to a file F, where $\mathcal{L}(G_F(\mathcal{R}_{Pgm}))$ is a sound classification. The write to F does not invalidate the soundness of $\mathcal{L}(F)$ provided that $\mathcal{L}(G_F(\mathcal{R}_{Pgm})) \preceq \mathcal{L}(F)$ holds, which is implied[9] by the following restriction.

> **MLFC Computation Restriction.** $\mathcal{L}(G_F(\mathcal{R}_{Pgm})) \preceq \mathcal{L}^{\sqcup}(\mathcal{R}_{Pgm})$ must hold for any value $G_F(\mathcal{R}_{Pgm})$ that Pgm computes and writes to a file F after reading a set $\mathcal{R}_{Pgm}$ of information items. □

MLFC Computation Restriction asserts that, given a set $\mathcal{R}_{Pgm}$ of inputs, Pgm does not compute and write a content unit that is more sensitive than the most sensitive input in $\mathcal{R}_{Pgm}$ or that includes a compartment not already covered by some input $\mathcal{R}_{Pgm}$. MLFC Write Restriction thus ensures that writing to some file F cannot make $\mathcal{L}(F)$ unsound (whereas writing a value with a label too high to satisfy MLFC Computation Restriction could make $\mathcal{L}(F)$ unsound because $\mathcal{L}(F)$ is a lower clearance than $\mathcal{L}(Pgm)$ and $\mathcal{L}(F) = \mathcal{L}(Pgm)$ holds).

Descriptions of multi-level confidentiality historically have ignored the calculations that programs undertake. In effect, MLFC Computation Restriction is assumed to be satisfied by all programs. That is not always a sound assumption. Potentially problematic cases include:

- *Functions whose outputs aggregate inputs.* For example, total combined force size available for deployment might well be more sensitive than the sizes of individual forces located at various staging areas (since the latter might be easily observed by the enemy).
- *Functions whose outputs are produced by analysis or logical deductions involving inputs.* For example, the output of an automated-trading system that recommends purchasing an equity might be deemed more sensitive (because knowledge of that output could affect other investor's actions) than the public knowledge about market trends that were inputs.

Note that MLFC Computation Restriction cannot be enforced by a reference monitor. Nor are there general purpose analysis methods to ascertain the classification of a function's outup given the classifications of its inputs. The insights and experience of a classification authority are needed for determining the sensitivity and the set of topics spanned by some computed value.

Defense Against Trojan Horse Attacks. Multi-level confidentiality resists *Trojan horse* attacks that DAC policies do not. A Trojan horse[10] is a program that appears useful but also implements hidden and nefarious functionality. A

[9] Here is a proof that restriction $\mathcal{L}(G_F(\mathcal{R}_{Pgm})) \preceq \mathcal{L}^{\sqcup}(\mathcal{R}_{Pgm})$ suffices and, therefore, $\mathcal{L}(G_F(\mathcal{R}_{Pgm})) \preceq \mathcal{L}(F)$ will hold: From (8.4) and MLFC Write Restriction $\mathcal{L}(Pgm) \preceq \mathcal{L}(F)$, we conclude by transitivity that $\mathcal{L}^{\sqcup}(\mathcal{R}_{Pgm}) \preceq \mathcal{L}(F)$ holds, and transitivity with MLFC Computation Restriction derives $\mathcal{L}(G_F(\mathcal{R}_{Pgm})) \preceq \mathcal{L}(F)$ (as desired).

[10] Greek mythology recounts how the 10-year Greek siege of Troy was ended by a clever subterfuge. The Greeks built a huge wooden horse, hid a small force of warriors inside, placed the horse outside the gates of Troy, and then appeared to abandon the siege by sailing out of sight. With the Greek force gone, the Trojans opened the city gates and moved the horse—thought to be a tribute marking the end of the siege—inside. The city residents celebrated.

game program, for example, might in the background (i) create a file that gives read privileges to the attacker and then (ii) copy confidential information to that file. DAC is useless at blocking this attack because all files being accessed by the game program are files that the invoker is authorized to access.

In comparison, that same Trojan horse would fail when run in an environment where Multi-level Document Confidentiality Policy is enforced. The MLFC Write Restriction, in particular, ensures that if an attacker has the clearance to read some file to which the Trojan horse has exfiltrated data then that attacker has sufficient clearance to read the files from which the data was copied.[11] So the attacker does not benefit from the Trojan horse.

Trusted Subjects. MLFC Write Restriction is violated by any program whose output files have less sensitive classifications than its input files. Yet write-downs are essential to the operation of many real systems. So the MLFC Write Restriction must sometimes be relaxed.

The classic example is a program $\texttt{encrypt}(k, txt)$ to encrypt an input txt using a key k. If a good encryption algorithm is employed then ciphertext $\texttt{encrypt}(k, txt)$ could be made public without revealing useful information about secret inputs k and txt. But requirement $\mathcal{L}(\texttt{encrypt}) \preceq \mathcal{L}(F_O)$ of MLFC Write Restriction is violated if the output of `encrypt` is written to a file F_O labeled with a sensitivity lower than the sensitivities labeling the files storing k and txt. Moreover, MLFC Read Restriction requires the sensitivity of $\mathcal{L}(\texttt{encrypt})$ to be at least that of the input files storing k and txt. The write-down to F_O, nevertheless, is safe.

Another useful form of safe write-down is exemplified by a service that reads files containing confidential information and writes status to an activity log file. The service must execute with a sufficient clearance to access the input files. If the content and timing of status updates to the activity log file do not reveal confidential information, then a high clearance should not be needed to read the activity log file. The service, however, violates MLFC Write Restriction if the activity log files are given a lower classification than the service has.

So there are compelling reasons for some programs to violate MLFC Write Restriction. Fortunately, MLFC Write Restriction is not the only way to prevent a given program from leaking information to principals that have lower clearances. An alternative is to analyze that program's code and certify that its logic ensures classified information will never be leaked.[12] Such programs

---

But once the sun had set, the Greek fleet turned around and headed back to Troy. At midnight, the Greek warriors inside the horse emerged, killed the Trojan guards, and opened the city gates. The Greek force, which by then had returned, entered the open gates and destroyed the city, thereby ending the war.

[11] If a copy F' of a file F is made by a Trojan horse TH then $\mathcal{L}(F) \preceq \mathcal{L}(TH)$ due to MLFC Read Restriction and $\mathcal{L}(TH) \preceq \mathcal{L}(F')$ due to MLFC Write Restriction. So, by transitivity, $\mathcal{L}(F) \preceq \mathcal{L}(F')$ holds. A clearance $\mathcal{L}(A)$ satisfying $\mathcal{L}(F') \preceq \mathcal{L}(A)$ is needed by an attacker A to read copy F'. But, by transitivity, $\mathcal{L}(F) \preceq \mathcal{L}(A)$ holds, so we have proved that clearance $\mathcal{L}(A)$ suffices for A to read F directly in all cases where A can read copy F'.

[12] Methods for such analysis are discussed in Chapter **??**.

can be exempted from the MLFC Write Restriction, and they are then called *trusted subjects*.

Trusted subjects provide an easy route for developers who must enforce the MLFC restrictions. However, a system that contains trusted subjects is only secure if its trusted subjects do not leak information. So prudence dictates that

- the system have few trusted subjects, and
- each trusted subject execute a small and simple piece of code, because large programs are difficult to analyze for information leaks.

Unfortunately, in practice, trusted subjects are too often designated indiscriminately and without (i) investing in the careful analysis required to rule out leakage or (ii) revisiting the system design to see if a restructuring might eliminate the need for a trusted subject in the first place.

Trusted subjects also can prove useful when labels are being assigned to coarse-grained objects but finer-grained access control is sought. For example, we might have an operating system that enforces Multi-level Document Confidentiality Policy on files, with each file storing a separate document. If we desire access control for portions of the file (corresponding to individual paragraphs), then we could employ a trusted subject. It would implement a reference monitor that controls reads and writes to individual paragraphs, according to classifications that are stored with each paragraph and clearances associated with principals that request the access. So, as before, the system is secure only if the trusted subject works as intended and does not leak the contents of paragraphs with high classifications to programs with lower clearances.

8.1.2 Multi-level Integrity

We might posit that the integrity of a file can be no better than the integrity of the inputs used in producing that file.[13] *Multi-level integrity* embodies this view, enforcing restrictions that prevent file corruption by any program whose inputs come from files and/or from arguments provided by a user who has been assigned a low clearance.

To start, assume that each user or file X is assigned a fixed *integrity label* $\mathcal{L}_I(X)$. We will consider $\mathcal{L}_I(X)$ *sound* if it gives a lower bound for the integrity that can be guaranteed for any output produced using information provided from X. Implicit in this soundness definition ("... gives a lower bound ...") is a partial order $\preceq_I$ on integrity labels. $\mathcal{L}_I(X) \preceq_I \mathcal{L}_I(X')$ holds if and only if element X has no better integrity than element X', where X might be a user (who provides arguments when invoking programs that produce outputs) or X might be a file. A MAC policy that prevents corruption of files is then:

[13] This view actually is rather conservative. Consider a program that outputs the value (if one exists) appearing as a majority of its inputs. When that majority of inputs have high integrity then the output has high integrity, despite the presence of other of inputs with low integrity from a minority. So the integrity of an output need not be inherited from the lowest-integrity input.

Multi-level File Integrity Policy. A program that is invoked by a user U and that reads from a file F' is authorized to update a file F only if $\mathcal{L}_I(F) \preceq_I \mathcal{L}_I(U)$ and $\mathcal{L}_I(F) \preceq_I \mathcal{L}_I(F')$ hold. □

That is, information written to a file F does not have lower integrity than what label $\mathcal{L}_I(F)$ promises.

Various schemes have been proposed for defining integrity labels. The classic one employs labels that resemble those used above for confidentiality. Here, labels are pairs $\langle T, C\rangle$, and partial order $\langle T, C\rangle \preceq_I \langle T', C'\rangle$ holds, as before, if and only if $T \leq T'$ and $C \subseteq C'$ hold.[14] But components T and C have different interpretations for integrity labels than they had for confidentiality labels.

- T signifies *trustworthiness*. It ranges over R (Random), P (Plausible), GR (Generally Reliable), and A (Authoritative), where R < P < GR < A.[15]
  - For files, T categorizes the extent to which the file contents are accurate and could not have been corrupted directly or indirectly by actions attackers instigate.
  - For users, T categorizes the competence and loyalty of the user, thereby indicating whether file integrity is likely to be compromised by user actions, such as invoking a program with the wrong arguments.
- C is a set of *compartments*. As with multi-level security, each compartment indicates specific topic areas.

A reference monitor enforcing Multi-level File Integrity Policy might have to block file writes. In particular, a program invoked with arguments provided by user U and that has read files $F_1, \ldots, F_n$ is permitted to update a file F only if the following holds.[16]

$$\mathcal{L}_I(F) \preceq_I \min(\mathcal{L}_I(U), \mathcal{L}_I(F_1), \ldots, \mathcal{L}_I(F_n)) \tag{8.5}$$

A naive implementation of this restriction would have the reference monitor record label $\mathcal{L}_I(F_i)$ whenever a file F_i is read during a program's execution. But

[14] As will become clear, integrity labels can be drawn from any set S with a partial order $\preceq_I$, provided (i) labels in S and $\preceq_I$ have a plausible interpretation in terms of some definition of integrity and (ii) if $S' \subseteq S$ holds then greatest lower bound $glb(S')$ is also a label in S. In the classic scheme, $glb(S')$ is defined as

$$glb(S') = \langle \min_{i \in S'}(T_i), \bigcap_{i \in S'} C_i \rangle.$$

for $S' = \{\langle T_1, C_1\rangle, \ldots \langle T_n, C_n\rangle\}$.

[15] Some authors employ labels that reinterpret sensitivities TS, S, C, U in terms of integrity. Others use the terms C (for Crucial), VI (for Very Important), and I (for Important)) that were used in Biba's original description of multi-level integrity, which incorporated a cost of consequences in each integrity label.

[16] The reason now becomes clear for requiring (in footnote 14) that $glb(S') \in S$ hold for every subset S' of set S of labels. By definition, if $S' \subseteq S$ holds then so does $(\forall s \in S' \colon glb(S') \preceq_I s)$. So from $glb(S') \in S$, we conclude there must exist a label for $\mathcal{L}_I(F)$ that satisfies (8.5) no matter what subset S' of labels are arguments to min.

the obligation to store labels for all files read can be (and typically is) eliminated by selecting a label $\mathcal{L}_I(Pgm)$ for the executing program Pgm. The following conditions, which together imply (8.5), are instead enforced by the reference monitor.

$$\mathcal{L}_I(F) \preceq_I \mathcal{L}_I(Pgm) \tag{8.6}$$

$$\mathcal{L}_I(Pgm) \preceq_I \mathcal{L}_I(U) \tag{8.7}$$

$$\mathcal{L}_I(Pgm) \preceq_I \mathcal{L}_I(F_i) \text{ for } 1 \le i \le n \tag{8.8}$$

In particular, (8.6) – (8.8) is each enforced by having the reference monitor (i) intercept file writes to prevent *write-up* attempts, (ii) intercept user invocation of programs, and (iii) intercept file reads to prevent *read-down* attempts:

> **MLFI Write Restriction.** $\mathcal{L}_I(F) \preceq_I \mathcal{L}_I(Pgm)$ must hold for a program Pgm to write a file F. □

> **MLFI Program Invocation Restriction.** $\mathcal{L}_I(Pgm) \preceq_I \mathcal{L}_I(U)$ must hold for an user U to invoke program Pgm. □

> **MLFI Read Restriction.** $\mathcal{L}_I(Pgm) \preceq_I \mathcal{L}_I(F)$ must hold for a program Pgm to read a file F. □

MLFI Write Restriction implies (8.6); MLFI Program Invocation Restriction implies (8.7); and MLFI Read Restriction implies (8.8).

Multi-level File Integrity with Dynamic Label Assignment. If integrity label $\mathcal{L}_I(X)$ is sound then any label L satisfying $L \preceq_I \mathcal{L}_I(X)$ will be sound for X, too. Thus, we can adopt the following rule to relax Tranquility Assumption (page 198) and allow changes to integrity labels during execution.

> **MLFI Label Reduction Restriction.** If $L \preceq_I \mathcal{L}_I(X)$ then label $\mathcal{L}_I(X)$ can be lowered to L prior to a write, program invocation, or read. □

This label reduction allows a program to proceed when it would have been blocked under a static label assignment. The following table summarizes the opportunities.

label reduced	MLFI rule potentially enabled
$\mathcal{L}_I(F)$	MLFI Write Restriction
$\mathcal{L}_I(Pgm)$	MLFI Program Invocation Restriction
$\mathcal{L}_I(Pgm)$	MLFI Read Restriction

When reduction of files labels is permitted, it is called an *object low-water mark* policy; when reduction in program labels is permitted, it is called a *subject low-water mark* policy.

Although reducing an integrity label could allow execution to continue, such a reduction can have insidious consequences. After integrity label $\mathcal{L}_I(F)$ on a file F is reduced, the integrity label of programs that read F might also

have to be reduced (to comply with MLFI Read Restriction). MLFI Write Restriction then necessitates reductions to integrity labels for files subsequently written by those programs. The process repeats, with integrity labels on files and programs spiraling ever lower. When execution finally terminates, output files would have extremely low integrity labels, offering minimal guarantees about the information stored in those files. Yet the integrity of output file contents might actually be quite high, with the weak integrity labels on those files merely artifacts of the MLFI restrictions and label reduction.

Multi-level Confidentiality and Integrity Combined. Multi-level File Confidentiality Policy (§8.1) offers no guarantees about data integrity. In fact, MLFC Write Restriction allows principals to write files they cannot read—so-called *blind writes*—and, therefore, an untrustworthy user can compromise the integrity of the most-secret files in the system. Multi-level File Integrity Policy, on the other hand, completely ignores confidentiality. MLFI Read Restriction authorizes an untrustworthy user to read files that are highly confidential.

Since confidentiality and integrity are often both important, we might contemplate a combined multi-level security policy. Each user U would receive a single clearance $\mathcal{L}_{CI}(U)$. A single clearance (rather than separate clearances $\mathcal{L}(U)$ and $\mathcal{L}_I(U)$) is sensible here, because background investigations give no basis to distinguish between a user's proclivity for compromising confidentiality versus corrupting integrity.

Because confidentiality and integrity of information are independent characteristics and enforced by different rules, it might seem natural in a combined policy to use two labels on each file F: $\mathcal{L}(F)$ for confidentiality and $\mathcal{L}_I(F)$ for integrity. In authorizing file reads and writes, we would impose the MLFC restrictions in connection with $\mathcal{L}(F)$ and MLFI restrictions in connection with $\mathcal{L}_I(F)$. So, for example, a read F is now permitted by a program Pgm provided $\mathcal{L}(F) \preceq \mathcal{L}(Pgm)$ holds (from MLFC Read Restriction) and $\mathcal{L}_I(Pgm) \preceq_I \mathcal{L}_I(F)$ holds (from MLFI Read Restriction).

Yet in practice, little is revealed by leaking low-integrity information because, by definition, low-integrity information can be inaccurate. So confidentiality is not independent of integrity, and therefore the expressive power of associating two labels with each file is unlikely to be helpful. That suggests using confidentiality label $\mathcal{L}(F)$ for each file F as the label $\mathcal{L}_{CI}(F)$ for a combined policy. However, this too turns out to be impractical, as we now see.

The authorization rules for the combined policy when each file F has a single label $\mathcal{L}_{CI}(F)$ are derived from the MLFC restrictions and the MLFI restrictions given above. For any program Pgm, not a trusted subject:

- Pgm is not authorized to read-up (MLFC Read Restriction) or write-down (MLFC Write Restriction).

- Pgm is not authorized to read-down (MLFI Write Restriction) or write-up (MLFI Read Restriction).

- And, because multi-level confidentiality does not support dynamic label assignment, labels on files cannot be changed.

Pgm is thus not authorized to read-up, read-down, write-up, or write-down! The combined policy is equivalent to using of a separate, isolated computer system for each different security label. We shouldn't be surprised that such a policy defends against information leaks and corruption, but the policy prohibits programs that read and write data having different security labels and, thus, is generally too restrictive to be practical.

8.2 Domain and Type Enforcement

With *domain and type enforcement* (DTE), an access matrix (rather than a partial order on labels) characterizes authorization for a MAC policy. Each entity that issues access requests is associated with a *domain*, which corresponds to a row in the access matrix; each object is associated with a *type*, which corresponds to a column in the access matrix. Cells of the access matrix give the set of privileges that members of domains are granted for objects of each given type. So an access matrix depicts a relation *Auth* comprising a set of triples $\langle D, O, p\rangle$, where $\langle D, O, p\rangle \in Auth$ holds if and only if entities in domain D have privilege p for objects of type T.

Figure 8.1 depicts an access matrix that grants to elements of domain D the set $\{p_1, p_2\}$ of privileges for objects of type T. Thus,

$$\langle D, T, p_1\rangle \in Auth \qquad\qquad \langle D, T, p_2\rangle \in Auth$$

are implied by this access matrix.

In DTE, domains are also considered types. This enables the access matrix to specify whether execution in one domain is authorized to transition into another (an event that likely occurs in conjunction with a control transfer). With domains considered types, the access matrix also is able to specify whether execution in one domain is authorized to issue a signal (a form of request) that interrupts execution of some other domain.

DTE imposes no restrictions on *Auth* and, therefore, can be used to implement a broad range of MAC policies. For example, to specify multi-level confidentiality (§8.1.1) using DTE:

- A distinct domain D is defined for all users and programs having the same label $\mathcal{L}(D)$.
- A distinct type T is defined for all files having the same label $\mathcal{L}(T)$.
- *Auth* is defined to include $\langle D, T, p\rangle$ if and only if the condition in the table below is satisfied, as prescribed by the applicable MLFC rule from §8.1.1.

$\langle D, T, p\rangle$	condition	MLFC rule
$\langle D, T, \texttt{invoke}\rangle$	$\mathcal{L}(D) \preceq \mathcal{L}(T)$	Program Invocation
$\langle D, T, \texttt{read}\rangle$	$\mathcal{L}(T) \preceq \mathcal{L}(D)$	Read Restriction
$\langle D, T, \texttt{write}\rangle$	$\mathcal{L}(D) \preceq \mathcal{L}(T)$	Write Restriction

domain	...	type T	...
⋮			
D		p_1, p_2	
⋮			

Figure 8.1: MAC Access Matrix

Multi-level integrity (§8.1.2) and any other policy involving labels and an ordering relation would be specified in an analogous way.

DTE can also be used to implement MAC policies that cannot be defined in terms of partial orders on labels. This makes DTE attractive for situations that multi-level security does not handle well. We illustrate with a simplified form of multi-level confidentiality involving labels S (secret) and P (public), with $\mathsf{P} \preceq \mathsf{S}$. Assume the usual rules for access to data by users and their applications—no "read-up" and no "write-down". Suppose, in addition, the system supports (i) an encryption service that inputs plaintext with label S and outputs ciphertext with level P and (ii) a decryption service that (given a suitable key) performs the inverse. A DTE authorization relation for this system can be specified using the access matrix in Figure 8.2, which employs the following domains and other types.

domain	type *Enc*	*Dec*	*ExecS*	*ExecP*	*FileS*	*FileP*
Enc	`invoke`		`invoke`		`read`	`write`
Dec		`invoke`	`invoke`		`read`, `write`	`read`
ExecS	`invoke`	`invoke`	`invoke`		`read`, `write`	`read`
ExecP				`invoke`		`read`, `write`

Figure 8.2: MAC for an Encryption/Decryption Service

Domains:

Enc	programs comprising the encryption service
Dec	programs comprising the decryption service
ExecS	users and application programs with clearance S
ExecP	users and application programs with clearance P

Other Types:

FileS	files containing data with classification S
FileP	files containing data with classification P

The access matrix of Figure 8.2 constrains execution in domains *ExecS* and *ExecP*—that is, execution of application programs—to comply with "no read-up" and "no write-down" as required by multi-level confidentiality. The access matrix also specifies that execution in *Enc* is authorized to read files of type *FileS* and to write files of type *FileP*[17] and thus *Enc* is being permitted to violate the mandates of multi-level confidentiality. The only way to allow *Enc* such access under the MLFC rules would require *Enc* to be a trusted subject, but that (in violation of the Principle of Least Privilege) unnecessarily grants *Enc* `write` for files in *FileS* and grants `read` for files in *FileP*.

DTE also avoids other problems that partial orders on labels cause for MAC policies. For instance, MLFC Computation Restriction (page 200) can be relaxed when DTE is used, so that a selected program is allowed to write content that is more sensitive than any of its inputs. With DTE, we can create a domain for that program and then assign privileges that allow writes to objects having higher-classification levels.

Trusted subjects are never needed with DTE because of the flexibility that access matrices provide. A domain can be assigned exactly the privileges needed to accomplish its tasks, compared to the unlimited privilege that a trusted subject receives. So, using DTE, a logging service could be authorized to write-down, although presumably this privilege assignment is made only after inspecting programs in that domain to ensure that they do not leak confidential information. The logging service domain is thus being trusted, but only in a limited way compared to a trusted subject. DTE also allows blind writes to be controlled, whereas multi-level security offers no solution.

The expressive power of DTE does have a cost, though. In order to use DTE, we must define a mapping from users and system objects to domains and types. And we must define *Auth*, either by enumerating its elements or by providing another means to decide whether $\langle D, T, p\rangle \in Auth$ holds, for every domain D, type T, and privilege p. The mapping to domains and types is not the problem. It has size linear in the number of users and system objects, which is comparable to the mapping to labels required when a partial order $\preceq$ (say) on labels is used for specifying authorization. But the enumeration of *Auth* can be significantly

[17] There are two cases to consider, depending on the value of $\mathcal{L}(Enc)$. If $\mathcal{L}(Enc) = \mathsf{S}$ holds then writes by *Enc* to files in *FileP* are authorized but would violate "no write-down", and if $\mathcal{L}(Enc) = \mathsf{P}$ holds then reads by *Enc* to files in *FileS* are authorized but would violate "no read-up". So some MLFC rule would be violated no matter what clearance is assigned to execution in domain *Enc*.

larger than the specification of partial order $\preceq$ that defines the exact same set of tuples $\langle D, T, p\rangle$. This is because *Auth* must explicitly enumerate all elements in the transitive closure of $\preceq$, whereas the specification of partial order $\preceq$ need not. Moreover, such an explicit enumeration might be the only option; some *Auth* relations cannot be specified using a partial order on domains and types because the relation is not transitive or is not antisymmetric.

8.3 MAC for Commerce

Accurate records about a company's assets and liabilities is essential for making short-term operational decisions as well as longer-term strategic ones.[18] Dishonest individuals left unchecked in this setting can perpetrate fraud by performing bogus updates to these records. Fraud, however, is as old as commerce itself. And the accounting profession long ago developed defenses, known as *financial controls*. They include:

- *Separation of Duty.* Tasks are made harder to subvert by requiring several individuals to participate. Collusion now becomes necessary in order to perpetrate fraud.
- *Prescribed Transformation Procedures.* Restrictions are imposed on who is authorized to update which records, how, and when. Considerably less damage is possible when only a small set of constrained programs is available for making updates.
- *Mandated Audit.* Each update is logged in a way that is immutable and identifies some individual as being accountable. Reviews are undertaken—periodically and randomly—to search logs for anomalies and to validate the accuracy and consistency of updated records. Would-be attackers are now deterred for fear of detection and because logs provide incriminating evidence for use in prosecution.

Notice that DAC's owner-control of authorization is incompatible with Separation of Duty. The multi-level security MAC policies described in §8.1 are not so useful here, either—they enforce authorization on individual reads and writes, whereas Separation of Duty and Transformation Procedures concern groupings of accesses. So the needs of commerce would seem to involve new kinds of MAC policies.

8.3.1 Separation of Duty

Fraud occurs when trusted individuals abuse their authority. Separation of duty defends against this by limiting the authority granted to any one individual.

[18] Assets and liabilities refer to all aspects of a company's current financial position: inventories of supplies, unpaid bills, accounts payable (including outstanding purchase orders and worker time-sheets), accounts receivable (including unpaid customer invoices), the cash disbursements journal (i.e., list of checks issued), the cash receipts journal (i.e., list of deposits), and so on.

With limited authority, a dishonest individual working alone only can cause limited damage; collusion by a set of dishonest individuals is now required to perpetrate a fraud. Care in hiring reduces the chances that a dishonest employee would find others who are both inclined to fraud and well positioned to collude.

With a *static* separation of duty, the authority given to each individual is fixed and pre-specified. This approach is often adopted by companies where job titles, which are linked to sets of tasks, bring the authority necessary to perform certain tasks but not others. For example, individuals in the Billing Department who generate customer invoices are authorized to update the accounts-receivable database but not authorized to update the warehouse-inventory database, whereas the opposite would hold for warehouse workers.

Static separation of duty has two significant shortcomings. First, attackers are able to identify individuals who, if recruited, could together commit a fraud. Second, it limits flexibility to reassign workloads when a business must cope with the unexpected (e.g., unusual demand or employee absences).

In *dynamic* separation of duty, the authority assigned to an individual is neither pre-determined nor fixed. The assignment might be random, or it might depend on state or history. A software development group, for example, might follow a code check-in regime that requires every module a programmer changes to be audited by a different programmer—a separation of duty for updates versus audits. If the auditor is selected based on who is the least overworked, then we have state-based dynamic separation of duty; if a programmer who has (or has not) previously contributed to the updated module must be selected, then we have history-based dynamic separation of duty.

Randomness can be used to good advantage in dynamic separation of duty. The absence of advance knowledge about who will participate in a given transaction frustrates[19] attempts by attackers to recruit collaborators. Financial institutions, for instance, require employees to be away from the office (at training or on vacation) annually, for an uninterrupted interval (typically one or two weeks); a randomly-selected peer is assigned the absent employee's workload. The required absence is, by design, long enough that the substitute would see irregularities indicative of fraud. And use of random selection makes it unlikely that the substitute would be in collusion with the absent employee.

Separation of duty might be tied to objects and/or tied to activities. In either case, enforcement could require that information about authority assignments be available and, if history is involved, that it be preserved for future reference.

- When separation of duty is tied to an object, then that object provides an obvious place for recording authority assignments. The code check-in example above concerns separation of duty associated with a module. The file storing the module's source code or that file's meta-data would be a natural place for recording the names of programmers and auditors involved in each a change.

[19] A collection of individuals is unlikely to collude unless they have come to trust each other (e.g., not to incriminate each other). It takes time to establish such trust. So attackers need long lead times to recruit collaborators.

- Activities, which typically are transient, generally do not offer places for long-term storage of information about authority assignments. Some activities (e.g., a process, thread, or transaction) are associated with system objects that could record authority assignments, but other activities (e.g., office workflows) require that new objects be created for this purpose. Objects that record authority assignments might have to be stored indefinitely if separation of duty is based on history. Garbage collection of these records now becomes an issue.

The enforcement of separation of duty for computer system operations that are initiated by a human user requires the system to authenticate that user. This works only if different user identifiers actually correspond to different humans. We are thus making assumptions about the ease with which the authentication protocol can be spoofed and about the care with which attributes are validated by the enrollment protocol when new users are registered.

Delegation of authority from one principal to another brings further complications, since distinct internal identifiers could now speak for the same principal. One option is to prohibit delegation from/to principals that are constrained by separation of duty. A second option is for each request to carry the identifier of the principal making the request as well as the identifiers of all principals that delegated authority to that requester. And a third option is to monitor delegations and create a central database that records equivalences that delegations induce among the identifiers being used.

The special case where one user delegates to another can be hard to handle. Such a delegation is a form of collusion if it is hidden from the enforcement mechanism. However, it is quite natural and often encouraged. One employee might lend a hand to another, or a manger might stand-in for a subordinate. A separation of duty policy can be formulated to support such delegations, provided these delegations are made known to the enforcement mechanism and, therefore, appropriate authority assignments still can be made.

Chinese Wall Policies. A *conflict of interest* occurs between two activities if an agent, in serving the best interests of one activity, cannot also further the best interests of the other. For example, a consultant advising one company about business strategy would have a conflict of interest by also advising a competitor—the consultant is an agent, the activity is advising, and knowledge of both competitor's strategies could enable improvements to one at the expense of the other.

Accepted business practice sometimes does allow competitors to be clients of individuals employed by a single enterprise[20] provided a *Chinese Wall* policy[21]

[20]However, the practice is by no means universally accepted. In the United States, investment banks do accept clients that are competitors, but law firms do not.

[21]This name alludes to the Great Wall of China, which was built to protect the northern border of China. The Great Wall of China is roughly 5500 miles long, comprising segments that are above-ground wall (earth, stones, and wood), trenches, and geologic barriers (rivers and mountains). The earliest segments were built in the 7th century B.C.E. These were later connected in the 3rd century B.C.E.

is enforced:

- An employee who is granted access to records provided by a client is not also granted access to records provided by competitors.

- An employee is barred from sending other employees any client records that the receiver is not authorized to access directly.

The first condition eliminates the conflict of interest for a single employee viewed as an agent, and the second condition rules out conflict of interest for the employer acting as a single agent that has competitors as clients.

The restriction on access to records by each employee is a separation of duty policy because it limits the authority given to each employee. Initially, an employee has accessed no records and thus can access records provided by any company. Thereafter, authorization to access records provided by a company depends on what other companies' records that employee has already accessed. The separation of duty is thus dynamic and based on history.

We formalize this separation of duty requirement by defining fixed[22] binary relation $\not\leftrightarrow$ on companies: $c' \not\leftrightarrow c''$ holds whenever companies c' and c'' are not competitors. We posit that no company competes with itself and that competition is mutual. Therefore, $\not\leftrightarrow$ is reflexive (i.e., $c' \not\leftrightarrow c'$ holds for all c') and symmetric (i.e., $c' \not\leftrightarrow c''$ holds if and only if $c'' \not\leftrightarrow c'$ does). But $\not\leftrightarrow$ is not transitive, because it would not be sound to conclude that c and c'' do not compete (i.e., $c \not\leftrightarrow c''$) from the knowledge that $c \not\leftrightarrow c'$ and $c' \not\leftrightarrow c''$ hold—an example is manufacturers c and c'' that compete with each other but not with their supplier c' and, thus, $c \not\leftrightarrow c'$ and $c' \not\leftrightarrow c''$ hold but $c \not\leftrightarrow c''$ does not.

Let $C_R(e)$ contain the name of every company that has provided records an employee e has seen. The restriction that employee e has not seen records provided by competitors is then implied by:

$$(\forall c', c'' \in C_R(e)\colon\ c' \not\leftrightarrow c'') \tag{8.9}$$

This security goal—for all employees e—must be enforced throughout execution.

To see records provided by a company c, an employee e invokes a read operation on a file associated with c. The read is authorized if $c \in C_R(e)$ already holds or if enlarging $C_R(e)$ to include c (as required so that $C_R(e)$ will satisfy its definition) does not falsify security goal (8.9).

CW Read Restriction. For an employee e to read a file associated with a company c

$$(\forall c' \in C_R(e)\colon\ c \not\leftrightarrow c') \tag{8.10}$$

must hold.[23] □

[22] Exercise 8.16 explores the case where, over time, competitors might change.

[23] To see formally that condition (8.10) in CW Read Restriction suffices for enforcing security

Writes allow an employee to transfer records from a file associated with one company to a file associated with a different company. So a write could cause some file associated with one company to contain records derived from information in files associated with another. No immediate harm is done if these companies are not competitors. The problems that could arise are illustrated in the following scenario.

1. e_1 reads a record from file f_1 associated with company c_1.
2. e_2 reads a record from file f_2 associated with company c_2.
3. e_2 writes to a file f_3 associated with company c_3.
4. e_1 reads a record from file f_3.

If companies c_1 and c_2 are competitors and if step 3 writes information e_2 read in step 2, then the read by e_1 in step 4 violates security goal (8.9).

A straightforward, though draconian, way to rule out such scenarios is simply to disallow writes that could transfer information from a file associated with one company to a file associated with another.

CW Write Restriction. For an employee e to write a file associated with a company c then $C_R(e) \subseteq \{c\}$ must hold.[24] □

---

goal (8.9), first note that the following is valid.

$$(\forall c', c'' \in C_R(e):\ c' \not\sim c'') \ \wedge\ (\forall c' \in C_R(e):\ c \not\sim c')$$
$$\Rightarrow$$
$$(\forall c', c'' \in (C_R(e) \cup \{c\}):\ c' \not\sim c'')$$

Moreover, both conjuncts in the first line are valid if a read were authorized—the first conjunct is security goal (8.9), which holds by construction, and the second conjunct is requirement (8.10) in CW Read Restriction. Since its antecedent is valid, the formula appearing as the conclusion of the implication will also be valid. That formula is security goal (8.9) with "$C_R(e)$" replaced by "$C_R(e) \cup \{c\}$". So we have demonstrated that security goal (8.9) holds for the value $C_R(e)$ has after e has read from a file associated with company c. Therefore, the read does not cause security goal (8.9) to be falsified, and we have demonstrated that the condition in CW Read Restriction is sufficient.

[24] To show formally that CW Write Restriction suffices to enforce security goal (8.9), let set $C_W(c)$ contain the name of every company providing information that currently might be in files associated with c.

We are safe in choosing $C_W(c) = \{c\}$ initially because, before any writes occur, information in files associated with c is, by definition, information that c provided. And after an employee e' writes to a file associated with c then $C_W(c)$ will continue to satisfy its definition if $C_W(c)$ is changed according to

$$C_W(c) := C_W(c) \cup C_R(e')$$

since $C_R(e')$ includes the set of companies that provided information that employee e' might include in its write. In addition, the assignment statement

$$C_R(e) := C_R(e) \cup C_W(c)$$

properly updates $C_R(e)$ after e has read from a file associated with c, because $C_W(c)$ (by definition) contains names of companies that provided information that might be found in that file.

In footnote 23 (page 213) justifying CW Read Restriction, we argued that security goal (8.9) had to hold with "$C_R(e)$" replaced by "$C_R(e) \cup \{c\}$". Now that we are using the assignment $C_R(e) := C_R(e) \cup C_W(c)$ to update $C_R(e)$, we conclude that "$C_R(e)$" should be replaced by

Condition $C_R(e) \subseteq \{c\}$ is not satisfied if e has read files associated with other companies besides c. So CW Write Restriction is allowing e to update a file associated with c only if e has read no files associated with other companies.

8.3.2 Transformation Procedures and Consistency

Obligations to maintain accuracy of information being stored by a company's computing system can be represented as *external consistency constraints*. These Boolean-valued formulas relate the state of a computing system to the state of its environment. For example, Widgets-R-Us might define the following external consistency constraint for its inventory database:

$$NW = w_S + \sum_{1 \le i \le N} w_i$$

This formula asserts that NW is an accurate count of the widgets that Widgets-R-Us owns—w_S denotes the number of widgets actually present in the Widgets-R-Us warehouse and w_i denotes the actual number of widgets being staged at workbench i of N in the Widgets-R-Us factory.

The environment—bank balances, warehouse contents, and so on—is likely beyond the control of a company's computing system which, after all, can only manipulate values stored in its database. But the environment is under the direct or indirect control of employees. We link the state of an environment to a computer system's database by mandating *business processes* that implement the following principle.

> **External Consistency Preservation.** Changes that could invalidate an external consistency constraint must involve participation by an employee, who is responsible for invoking a program that performs corresponding updates to the computing system's state. □

Assurance about the truth of an external consistency constraint can be established by people performing audits. The auditors first gather evidence about the state of the environment: query the bank, inventory the warehouse, check with creditors to confirm accounts payable, check with customers to confirm accounts receivable, and so on. Then the auditors compare and reconcile any differences between the evidence they have and the information being stored by the computer system.

Trustworthy employees perform their jobs properly, which means they undertake mandated updates but no others. Untrustworthy employees, however, might initiate bogus updates for reasons ranging from incompetence to malevolence. We help defend against such fraud by instantiating the Principle of Least Privilege and restricting what updates each employee is authorized to perform.

"$C_R(e) \cup C_W(c)$" after e reads a file associated with c. If $C_W(c) = \{c\}$ holds then $C_R(e) \cup \{c\}$ and $C_R(e) \cup C_W(c)$ are equal. Moreover, $C_W(c) = \{c\}$ holds initially and subsequent changes to $C_W(c)$ are characterized by $C_W(c) := C_W(c) \cup C_R(e')$. So it suffices to require that $C_R(e')$ in that assignment statement satisfy $C_R(e') \subseteq \{c\}$ whenever an employee e' writes to a file associated with c. That condition is exactly what CW Write restriction requires.

- A predefined set of *transformation procedures* is defined. Each transformation procedure definition

  $$\begin{array}{l} tp\text{: } pgm(a_1, \ldots, a_n) \\ \quad \textbf{access } F_1{:}\{op_1^1, \ldots op_1^{m_1}\},\ \ldots,\ F_p{:}\{op_p^1, \ldots op_p^{m_p}\} \end{array}$$

  specifies a program *pgm* and argument list $a_1, \ldots, a_n$, where *pgm* has been certified to

  (i) implement some mandated step of a business process, and

  (ii) log all updates it performs.

  The transformation procedure definition authorizes *pgm* to access files F_1, ..., F_p, where accesses to each file F_i may use only operations op_i^1, ... $op_i^{m_i}$.

- A reference monitor is deployed in conjunction with an authorization relation *EmpAuth*. The reference monitor ensures that employee e can invoke transformation procedure *tp* only if $\langle e, tp \rangle \in EmpAuth$ holds.

These restrictions on updates to information stored in the computer system could be easily circumvented if employees were able to modify *EmpAuth* or make changes to the set of transformation procedures. We mitigate against that by imposing a separation of duty policy.

> **Independent Certification.** No employee who is authorized to execute transformation procedures is authorized to modify *EmpAuth* or to participate in certifying or installing programs invoked by any transformation procedure. □

Collusion is now necessary before an attacker can get a bogus transformation procedure installed.

Double-Entry Bookkeeping. The state of a computer system can be checked to reveal whether stored values are unreasonable in isolation or in comparison to other stored values.[25] An *internal consistency constraint* is a Boolean-valued formula that is satisfied if the computer system state does not exhibit certain pre-specified anomalies. Stronger (i.e., more restrictive) internal consistency constraints are obviously better, because they are satisfied by fewer anomalous states. Internal consistency constraints that are too strong might, however, erroneously disqualify states. For example, an internal consistency constraint is not very useful if it can be falsified by executing a transformation procedure whose invocation is mandated by some business process. The ideal internal consistency constraint would be falsified only by invocations of transformation procedures that are not mandated by some business process.[26]

[25] But an outside audit is the only way to determine whether external consistency constraints are satisfied.

[26] In prohibiting certain invocations, an ideal internal consistency constraint also prohibits fraudulent acts of omission. Suppose some mandated transformation procedure should be

Set	Class	Informal Description
$\mathcal{A}$	assets	things that add value to the business
$\mathcal{E}$	expenses	expenditures over some specified period
$\mathcal{I}$	income	income over that same period
$\mathcal{L}$	liabilities	things that reduce the value of the business
$\mathcal{Q}$	equity	overall value of the business

Figure 8.3: Classes of Accounts in Double-entry Bookkeeping

In *double-entry bookkeeping*, a single internal consistency constraint is defined over the set of *accounts* that collectively represent the state of a business. Every account stores an initial balance and a sequence of *postings*, where each posting describes an increment or decrement to that account's balance. The *accounting equation*

$$\sum_{i \in \mathcal{Q}} i \quad = \quad \sum_{i \in \mathcal{A} \cup \mathcal{I}} i \quad - \quad \sum_{i \in \mathcal{L} \cup \mathcal{E}} i \tag{8.11}$$

is a consistency constraint, with Figure 8.3 defining categories $\mathcal{A}$, $\mathcal{E}$, $\mathcal{I}$, $\mathcal{L}$, and $\mathcal{Q}$ that accountants traditionally use to group accounts. Notice, any single (non-zero) posting to a only one account will falsify (8.11). The need to make two or more postings is what leads to the name double-entry bookkeeping.

Under double-entry bookkeeping, the entire set of postings that result from executing a transformation procedure must not invalidate (8.11). Moreover, where needed, a sequence $tp_1\ tp_2\ \ldots\ tp_n$ can be enforced on the steps mandated by some business process. To implement such sequencing, each transformation procedure tp_i is defined to fail if some prespecified Boolean condition $pre(tp_i)$ is *false* when tp_i is invoked. Condition $pre(tp_i)$ will typically establish that some value is present in certain specified set of accounts; execution of tp_i then posts decrements and increments to accounts in a way that falsifies $pre(tp_i)$ but makes $pre(tp_{i+1})$ *true*, which implies that tp_{i+1} must be performed next. So value in designated accounts serves as baton that gets passed from one transformation procedure in a sequence to the next one in that sequence.

To make this concrete, consider the transformation procedures *purchase*, *orderRcvd*, and *invRcvd* sketched in Figure 8.4. Each is specified using the syntax discussed earlier in this section (see page 215), followed by the program the transformation procedure executes. Those programs use two new statement types: **post** and **check**.

A **post** statement

post $\langle +cost, id \rangle$ **to** $AcntPay_{\mathcal{L}}$

causes element $\langle +cost, id \rangle$ to be appended to the sequence of postings associated with account $AcntPay_{\mathcal{L}}$. Identifier id enables this posting to be linked with other

invoked, but isn't. Sooner of later, some transformation procedure will be invoked. That invocation, by definition, is premature—so it is not (yet) mandated and thus will falsify an ideal internal consistency constraint.

-

purchase:*GenPO*(*item*, *supplier*, *cost*, *id*)
access $AcntPay_{\mathcal{L}}$:{**post**},
$ShpExp_{\mathcal{A}}$:{**post**}

GenPO: **program**(*item*, *supplier*, *cost*, *id*)
post $\langle +cost, id\rangle$ **to** $AcntPay_{\mathcal{L}}$
post $\langle +cost, id\rangle$ **to** $ShpExp_{\mathcal{A}}$
send order number *id* **for** *item* **to** *supplier*
end *GenPO*

orderRcvd:*XferDeliv*(*id*)
access $ShpExp_{\mathcal{A}}$:{**post**, *read*},
$Invtry_{\mathcal{A}}$:{**post**, *read*}

XferDeliv: **program**(*id*)
check $\langle \hat{c}, id\rangle \in ShpExp_{\mathcal{A}} \wedge \langle \hat{c}, id\rangle \notin Invtry_{\mathcal{A}}$ **then**
post $\langle -\hat{c}, id\rangle$ **to** $ShpExp_{\mathcal{A}}$
post $\langle +\hat{c}, id\rangle$ **to** $Invtry_{\mathcal{A}}$
else exception("Unsolicited or duplicate delivery")
end *XferDeliv*

invRcvd:*GenCheck*(*supplier*, *cost*, *id*, *chkNo*)
access $Invtry_{\mathcal{A}}$:{*read*},
$AcntPay_{\mathcal{L}}$:{**post**, *read*}
$ChkAcnt_{\mathcal{A}}$:{**post**}

GenCheck: **program**(*supplier*, *cost*, *id*, *chkNo*)
check $\langle +cost, id\rangle \in AcntPay_{\mathcal{L}}$
$\wedge \langle +cost, id\rangle \in Invtry_{\mathcal{A}}$
$\wedge \langle -cost, id\rangle \notin AcntPay_{\mathcal{L}}$
then
post $\langle -cost, id\rangle$ **to** $AcntPay_{\mathcal{L}}$
post $\langle -cost, chkNo\rangle$ **to** $ChkAcnt_{\mathcal{A}}$
send check *chkNo* **for** *cost* **to** *supplier* re invoice *id*
else exception("Not delivered, not ordered, or paid once")
end *GenCheck*

Figure 8.4: Inventory Purchase Steps

postings. We adopt the convention that account names include a subscript to indicate a class from Figure 8.3, so it becomes easy to see that no transformation procedure falsifies accounting equation (8.11).

A **check** statement either executes a **then** clause or an **else** clause, depending on whether some given Boolean condition evaluates to *true*. That Boolean condition can introduce new identifiers, which subsequently are referenced within the scope of the **then** and have initial values satisfying the Boolean condition. For example, in

$$\textbf{check } \langle \hat{c}, id \rangle \in \mathit{ShpExp}_{\mathcal{A}} \ \wedge \ \langle \hat{c}, id \rangle \notin \mathit{Invtry}_{\mathcal{A}} \textbf{ then } ...$$

(taken from *XferDeliv* in Figure 8.4), variables $\hat{c}$ and id when execution of the **then** starts have initial values that correspond to some prior posting to $\mathit{ShpExp}_{\mathcal{A}}$ that has not also been made to $\mathit{Invtry}_{\mathcal{A}}$.

The transformation procedures in Figure 8.4 together implement the standard business process for inventory acquisition, under the assumption that authorization relation *EmpAuth* enforces a separation of duty policy that restricts each employee to invoking at most one of the three transformation procedures. The sequence of steps for inventory acquisition are:

1. An authorized employee in the front office invokes *purchase* to issue a purchase order. This transformation procedure executes program *GenPO* with authorization for posting to accounts $\mathit{AcntPay}_{\mathcal{L}}$ and $\mathit{ShpExp}_{\mathcal{A}}$ (only). A *purchase* creates a liability (because the goods will ultimately have to be paid for), which is reflected by the posting to $\mathit{AcntPay}_{\mathcal{L}}$. The expectation for the delivery is an asset, reflected in the posting to $\mathit{ShpExp}_{\mathcal{A}}$. Identifier id is thereafter associated with this purchase order and is expected to appear in the paperwork that accompanies the delivery and in the supplier's invoice.

2. An authorized employee on the loading dock invokes *orderRcvd* when delivered goods are accompanied by a packing slip that references identifier id. This transformation procedure executes program *XferDeliv* with authorization for reading and posting to accounts $\mathit{ShpExp}_{\mathcal{A}}$ and $\mathit{Invtry}_{\mathcal{A}}$. Execution checks $\mathit{ShpExp}_{\mathcal{A}}$ to ensure the delivery was expected and checks $\mathit{Invtry}_{\mathcal{A}}$ to make sure the delivery is not a duplicate. A decrement is then posted to $\mathit{ShpExp}_{\mathcal{A}}$ because the shipment is no longer expected, and an increment is posted to $\mathit{Invtry}_{\mathcal{A}}$ because inventory has been increased.

3. An authorized employee in the payments department invokes *invRcvd* to pay the supplier when an invoice is received for goods with identifier id. This transformation procedure executes program *GenCheck* with authorization for reading $\mathit{Invtry}_{\mathcal{A}}$, for posting to and reading $\mathit{AcntPay}_{\mathcal{L}}$, and for posting to $\mathit{ChkAcnt}_{\mathcal{A}}$. The transformation procedure validates that a purchase order was issued (by checking $\langle +cost, id \rangle \in \mathit{AcntPay}_{\mathcal{L}}$), the delivery has been added to inventory (by checking $\langle +cost, id \rangle \in \mathit{Invtry}_{\mathcal{A}}$), and the invoice has not yet been paid (by checking $\langle -cost, id \rangle \notin \mathit{AcntPay}_{\mathcal{L}}$).

A posting to $AcntPay_{\mathcal{L}}$ then cancels the liability of an unpaid invoice; a posting to $ChkAcnt_{\mathcal{A}}$ records an equivalent expense.

Notice how *orderRcvd* is enabled (due to the **check** that starts *XferDeliv*) by a posting to $ShpExp_{\mathcal{A}}$ not also appearing in $Invtry_{\mathcal{A}}$. And *invRcvd* is enabled (due to the **check** that starts *GenCheck*) by postings to $AcntPay_{\mathcal{L}}$ and $Invtry_{\mathcal{A}}$ not also appearing in $AcntPay_{\mathcal{L}}$. So a sequence—*purchase*, *XferDeliv*, *invRcvd*—is imposed on the execution order for transformation procedures that all concern the same value of *id*.

Also note each of the transformation procedures in Figure 8.4 is enabled (by a posting some other transformation procedure makes) or has its postings checked by another transformation procedure. Therefore, (at least) two separate transformation procedures are always involved in any update to an account. Since we are assuming that no employee is authorized by *EmpAuth* to invoke more than one transformation procedure, the design of these transformation procedures implies that two or more employees must collude in order to embezzle funds or inventory.

8.4 Role-based Access Control

Long-lived enterprises and institutions tend to be structured around *roles*—not around individuals. A role might be identified with a particular job title, a project, a client, or some combination. Each role grants privileges. The privileges authorize only those actions expected from a role's occupants. This set of actions would be relatively fixed, so the set of privileges associated with a given role tends to be static. Authorization schemes we have discussed thus far, which decide access based on user identity, obviously could implement such a policy. But since a user's roles tend to change over time, the administration of privilege-assignments to users based on identity is cumbersome. So identity-based schemes are not well suited for institutional settings; roles are a more suitable basis for authorization.

With *role-based access control* (RBAC), access requests are made during *sessions*. Each session S speaks for a set $\rho(S)$ of *roles* occupied by the user $\mu(S)$ who instigates the session. A system typically provides commands to instigate or terminate a session and, from within a session, to enter or exit occupancy in a specified role. For example, a graduate student in Computer Science, after being authenticated as user EK, might begin a session S (say) and then enter role studentCS6110 followed by role graderCS5430: $\mu(S)$ = EK and $\rho(S)$ = {studentCS6110, graderCS5430}.

The privileges associated with a session are constrained by two relations.

- $UserRoles(U)$ is the set of roles that a user U is authorized to occupy.
- $RolePrivs(R)$ is the set of privileges granted by a role R.

Throughout every session S, *UserRoles* restricts $\rho(S)$ based on $\mu(S)$:

$$\rho(S) \subseteq UserRoles(\mu(S))$$

RolePrivs is used along with $\rho(S)$ to authorize requests made within a session S. For example, *RolePrivs*(studentCS6110) might include privileges granting read access to lecture notes (but not homework solution sets) for course CS6110, whereas *RolePrivs*(graderCS5430) might include privileges that grant write access to lecture notes and read access to solutions sets for CS5430.

RBAC is agnostic about the kinds of operations that privileges authorize. Privileges might authorize low-level operations, such as reads or writes to particular files. Or they might authorize high-level operations that bundle access by some given program to specific objects, as required by transformation procedures (§8.3.2).

Role Hierarchy and Privilege Inheritance. Roles in an organization often inherit privileges from other roles. For example, officers of a club inherit all privileges the member role grants. Inheritance causes the inferior role to add its privileges to the privileges that are associated with the superior role. RBAC provides the *role inheritance* relation $\sqsubset$ to specify that occupants of one role also are granted all of the privileges associated with some inferior role.

$R \sqsubset R'$ asserts that an occupant U of role R' also receives all privileges granted by role R—whether or not U is an occupant of role R. This can be formalized in terms of a set $\rho^*(S)$ characterizing roles that a user $\mu(S)$ in effect occupies by executing enter commands in session S or through role inheritance:

$$\rho^*(S) \;=\; \rho(S) \cup \{R' \mid R \in \rho(S) \wedge R' \sqsubset R\}$$

Set $privs(S)$ of privileges that a session S grants to $\mu(S)$ is:

$$privs(S) = \bigcup_{R \in \rho^*(S)} RolePrivs(R) \tag{8.12}$$

For example, an RBAC scheme at Cornell University might specify role inheritance relations

$$\mathsf{cornellian} \sqsubset \mathsf{CUstudent} \qquad \text{and} \qquad \mathsf{cornellian} \sqsubset \mathsf{CUstaff}$$

to assert that occupants of the CUstaff and CUstudent roles also receive privileges (e.g., access to the library, parking, and cafeterias) granted to occupants of the cornellian role (a label for members of the university community), even though *RolePrivs*(CUstaff) and *RolePrivs*(CUstudent) do not explicitly list the privileges in *RolePrivs*(cornellian).

Not only might multiple distinct roles inherit privileges from a single role, but a single role might inherit privileges from multiple roles. Managers are often empowered to do anything their immediate subordinates can. If emp_1, ..., emp_n are the roles occupied by individuals supervised by the occupant of role mngr, then specifying role inheritance relations

$$\mathsf{emp}_1 \sqsubset \mathsf{mngr}, \;\; \ldots, \;\; \mathsf{emp}_n \sqsubset \mathsf{mngr}$$

would ensure that mngr occupants are granted the needed privileges.

Term	Informal Description
$\rho(S)$	set of roles entered in session S
$\rho^*(S)$	set of roles effectively occupied in session S
$\mu(S)$	user who instigated session S
$\mathcal{A}$	set of all sessions currently active
$privs(S)$	set of privileges granted in session S
$R \sqsubset R'$	role R' inherits privileges granted by role R
$Roles$	set of all roles
$RolePrivs(R)$	the set of privileges granted by role R
$Users$	set of all users
$UserRoles(U)$	set of roles user U is authorized to occupy

Figure 8.5: Terms for Formulating RBAC Constraints

Role inheritance relation $\sqsubset$ is not strictly necessary for defining the set of privileges associated with a session. We would obtain the same set $privs(S)$ of privileges for each session S simply by augmenting *RolePrivs* based on the role inheritance relations. The implicit granting of privilege through role inheritance is valuable for system administration, though. When $R \sqsubset R'$ holds, a change to $RolePrivs(R)$ not only changes what privileges are granted to occupants of role R but automatically changes what privileges are granted to occupants of all roles R' satisfying $R \sqsubset R'$. The same effect is impossible to achieve by analyzing and updating the $RolePrivs(R')$ sets. Without the specification of all role inheritance relations, a system administrator cannot infer from $RolePrivs(R')$ whether role R' satisfies $R \sqsubset R'$. Yes, the administrator could check whether R' satisfies $RolePrivs(R) \subseteq RolePrivs(R')$. But $RolePrivs(R) \subseteq RolePrivs(R')$ might hold either because $R \sqsubset R'$ holds or because roles R and R' coincidentally authorize overlapping privileges. And without knowing whether a role inheritance relation $R \sqsubset R'$ is intended, a system administrator cannot know whether to change $RolePrivs(R')$ when changes to $RolePrivs(R)$ are made.

Constraints. Separation of duty and other policies that restrict mutual occupancy of roles are specified within RBAC by giving *constraints*. Each constraint is a Boolean expression formulated using the terms given in Figure 8.5. An RBAC implementation is expected to block any action that, if allowed to proceed, would invalidate any of the constraints.

The constraint to specify that any user authorized to occupy role R is not authorized to occupy R' and *vice versa* would be:

$$(\forall U \in Users\colon\ R \notin UserRoles(U) \lor R' \notin UserRoles(U)) \tag{8.13}$$

A weaker separation of duty policy is defined by constraint

$$(\forall S \in \mathcal{A}\colon\ R \notin \rho(S) \lor R' \notin \rho(S)) \tag{8.14}$$

since it excludes users from occupying roles R and R' within a single session but does not prevent a user from occupying roles R and R' in separate, concurrent

sessions; the following constraint does.

$$(\forall S, S' \in \mathcal{A}\colon\ \mu(S) = \mu(S')\ \ \Rightarrow\ \ (R \notin \rho(S) \vee R' \notin \rho(S'))) \tag{8.15}$$

Notice, however, that (8.15) achieves the desired effect only if $\mu(S) \neq \mu(S')$ implies that sessions S and S' are instigated by different individuals. The soundness of that assumption depends both on the means by which individuals are authenticated and on the enrollment protocol employed to register new users (where each individual's identify presumably would have been authenticated).

Role inheritance brings an additional complication for separation of duty policies. Because a user occupying one role now could also be granted privileges from another role, separation of duty policies for occupancy of roles do not necessarily impose separation of duty for granting of privileges. Separation of duty for privileges can be achieved by adding constraints, though. For example, we specify that a privilege p is granted to only a single user at any time (without identifying what user or restricting which privileges various roles grant) with the following constraint.

$$\begin{aligned}\neg(\exists S, S' \in \mathcal{A}\colon\quad & \mu(S) \neq \mu(S')\ \wedge\ R \in \rho^*(S)\ \wedge\ R' \in \rho^*(S') \\ & \wedge\ p \in \mathit{RolePrivs}(R)\ \wedge\ p \in \mathit{RolePrivs}(R'))\end{aligned} \tag{8.16}$$

Besides specifying separation of duty policies, constraints can offer a way to incorporate user attributes or system state into access control decisions. The following constraint restricts occupancy in a role R to occur during normal working hours

$$(\forall S \in \mathcal{A}\colon\ R \in \rho(S)\quad \Rightarrow\ \ 0900 \leq \mathit{time} \leq 1700) \tag{8.17}$$

if variable *time* evaluates to the current time. And given a function $\mathit{locate}(U)$ that evaluates to the current location of a user U, the following constraint restricts occupancy in R to those users working at the office (versus, say, at home or at an Internet cafe).

$$(\forall S \in \mathcal{A}\colon\ R \in \rho(S)\ \ \Rightarrow\ \ \mathit{locate}(\mu(S)) = \text{office}) \tag{8.18}$$

Easily specified does not mean easily enforced. RBAC constraints are typically enforced by using a reference monitor. This reference monitor must intercept and block all events that, if allowed to proceed, would falsify any constraint. Enforcement of a constraint now depends on the feasibility of intercepting relevant events. Certain events are cheap to intercept because they involve execution within the operating system:

- system administrator commands to change the sets of users or roles, the roles each user is authorized to occupy, the privileges associated with each role, and the role inheritance relation;
- system calls that allow a user to instigate/terminate a session or to enter/exit occupancy in a specified role within a session.

Moreover, we might reasonably expect that these events are the sole means for causing the terms in Figure 8.5 to change value at run-time.

So a reference monitor incorporated into the operating system can be used to support constraints (8.13) – (8.16). Enforcement of a constraint becomes problematic if it depends on functions (e.g., *time* and *locate*$(\cdot)$) that change value undetectably or too frequently for the the reference monitor to be invoked. Although (8.17) can be enforced by scheduling timer-interrupts that invoke the reference monitor at 0900 and 1700, other constraints involving *time* might require more frequent checking than would be practical. And enforcement of constraints involving location might well be completely infeasible.

Exercises for Chapter 8

8.1 Instead of using pairs, we might define a label L to be a set $\{L_1, L_2, \ldots, L_n\}$ of pairs, where each $L_i = \langle S_i, C_i \rangle$ comprises a sensitivity S_i and a (single) compartment C_i.

(a) Define an ordering relation on this new kind of label and argue that your proposal is sensible.

(b) Compare the expressive power of this new kind of label with the labels DoD uses, assuming that the same sets of sensitivity labels and compartments are used in both schemes.

8.2 Two kinds of processes access a database that is stored in a file F. A *reader* process issues reads; a *writer* process issues both reads and writes. Let R_1, R_2, ..., R_n denote the reader processes and let W_1, W_2, ..., W_n denote the writer processes. The readers and writers are disjoint, and the MLC restrictions for Multi-level File Confidentiality are enforced. What properties must be satisfied by classification labels for F, the reader processes, and the writer processes if we wish to enforce:

- readers cannot convey content to writers or other readers, and
- writers can convey content to readers and to writers.

Justify the need for these restrictions.

8.3 `AppropriateTube` is building software to enable creation and dissemination of videos for children. Two pieces of meta-data characterize who may view each video: Age, measured as years since birth, defines the minimum age of an appropriate viewer; content is summarized by a set containing some of the following content-descriptors:

> Alcohol, Bambi, BarbieAndKen, Barney, Disrespect, Evolution, IntelligentDesign, Sexuality, TeddyBears, VerbalAbuse, Violence.

The system envisaged by `AppropriateTube` will work as follows.

- A web site (`www.NoOffense.com`) will store videos that users contribute. Meta-data that is stored for each video gives an age and content-descriptors. The age is the minimum recommended age for viewers; the content-descriptors summarize what the video contains.
- A browser app will enable parents to upload new videos. The app queries the user for age- and content-descriptor meta-data. Parents are assumed to be truthful when answering these queries.
- A combine-videos browser app will enable users to create a mash-up of videos already stored by `www.NoOffense.com`; that mash-up is then stored at `www.NoOffense.com`. The app automatically generates age and content-descriptor meta-data for the mash-up and stores that information along with this new video.
- A video-viewer browser app will allow users to watch videos stored by `www.NoOffense.com`. The app reads a configuration file stored in the home directory of the user (presumed to be a child or adult) who is running the app. This configuration file gives the birth year of the user and gives a list of content-descriptors for videos the user is allowed to see. The app displays only those videos that are age-appropriate and belief-appropriate for the use.

(a) Give rules for how the meta-data for a mash-up should be produced by the combine-videos app.

(b) Give rules for how the meta-data for each video should be used by the video-viewer app.

8.4 Sometimes individual data are less sensitive than their aggregations. Suppose that a *database* comprises a number of *datasets*. Further, suppose that each dataset and database D has a label $\mathcal{L}(D)$ that ranges over traditional sensitivity labels U, C, S, or TS. So we might have $\mathcal{L}(A) = \mathsf{U}$ and $\mathcal{L}(B) = \mathsf{U}$ for datasets A and B but $\mathcal{L}(R) = \mathsf{S}$ for an aggregation R derived from A and B. We write $R = \{A, B\}$ to specify that an aggregation R is derived from A and B, where we expect:

$$\mathcal{L}(\{A\}) = \mathcal{L}(A) \quad \text{for every dataset } A$$
$$\mathcal{L}(R) \subseteq \mathcal{L}(R') \quad \text{if } R \subseteq R'$$

(a) Give a real-world example of a fixed collection of datasets and aggregations in which some aggregation is more sensitive than any of its constituents.

(b) Suppose a program performs (i) reads on datasets and aggregations, and (ii) reads and writes on a fixed set of *objects*, where $\mathcal{L}(Obj)$ denotes the fixed label assigned to object Obj. Exhibit a sequence of reads and writes involving some set of objects along with the datasets and aggregates you give in (a). The sequence should violate the policy

> Label $\mathcal{L}(Obj)$ on each object Obj is not less sensitive than the information Obj

even though all reads and writes satisfy MLFC Read Restriction and MLFC Write Restriction.

(c) Suppose labels are sets of names for datasets and objects instead of being distinguished names U, C, S, or TS. Give restrictions on reads and writes (analogous to MLFC Read Restriction and MLFC Write Restriction) that enforce the following policy.

> Information written into an object Obj must be derived from datasets and objects having names that appear in $\mathcal{L}(Obj)$.

8.5 Multi-level Confidentiality for Computers (§8.1.1) assumes that a read or write operation will be blocked if it violates MLFC Read Restriction or MLFC Write Restriction. Suppose, instead, that such an access attempt returns an error message, and assume that an attempt to access a file that does not exist returns the same message. We wish to support an additional operation

```
createFile(FName, Lab)
```

whose execution creates a new empty file that is named `FName` and has label `Lab`.

(a) What, if any, rules about labels should be imposed on execution of reads, writes, and `createFile` to ensure that classified information cannot be leaked?

(b) Suggest error messages to be returned by `createFile` for the following two cases: (i) the rule suggested in (a) is not satisfied, (ii) the rule suggested in (a) is satisfied but a file named `FName` already exists. Give a rationale for the wordings you propose.

8.6 A new scheme has been proposed for implementing Multi-level Confidentiality. In this scheme, labels on files and programs may be changed at any time according to the following rules.

> **File Label Change.** Label $\mathcal{L}(F)$ on a file F (i) can be increased or (ii) can be decreased to the largest label on any item that has been written so far to that file. □

> **Program Label Change.** Label $\mathcal{L}(Pgm)$ on a program Pgm (i) can be increased or (ii) can be decreased to the largest label on any item that has been read so far by that program. □

Instead of MLFC Read Restriction and MLFC Write Restriction, reads and writes are governed by:

- If $\mathcal{L}(F) \leq \mathcal{L}(Pgm)$ holds then a write to file F by Pgm is a no-op rather than causing execution of Pgm to be blocked or terminated; otherwise the write is allowed to proceed.
- If $\mathcal{L}(Pgm) \leq \mathcal{L}(F)$ holds then a read to file F by Pgm returns the error message "file unavailable" instead of returning the contents of F; otherwise the read is allowed to proceed.

Is it possible for Pgm to learn whether the contents of a file F satisfy some given predicate, even though $\mathcal{L}(Pgm) \leq \mathcal{L}(F)$ holds? If so, give a scenario—the files, their labels, and the sequence of reads and writes that constitutes the attack; if not, give an argument that explains why the information cannot be learned by Pgm.

8.7 Suppose the MLFC Write Restriction were replaced by:

> **Revised MLFC Write Restriction.** $\mathcal{L}(Pgm) = \mathcal{L}(F)$ must hold for a program Pgm to write into a file F information that Pgm has read. □

A program now can read anything it writes, so the program can confirm that its writes are successful. However, some writes that would be permitted by MLFC Write Restriction—namely, writes to files that have higher classifications—are no longer be permitted. Can a program that satisfies MLFC Write Restriction be compiled mechanically to a program or set programs that together (i) satisfy Revised MLFC Write Restriction and (ii) exhibits no loss of security?

8.8 Let G_F be a function whose output to file F is inherently more sensitive than the classification label assigned to any file it reads. Comment on the likely rationale for satisfying MLFC Computation Restriction by using each of the following strategies. Also comment on the long-term practicality of the proposed approach.

(a) Require that files read by G_F have classification labels that are higher than their content warrants.

(b) Require that G_F read and ignore input from an additional file that has a classification at the same level as the output that will be produced by G_F.

8.9 The restrictions in §8.1.1 for confidentiality and §8.1.2 for integrity are formulated for a single monolithic program. Systems typically comprise multiple programs, where one program might call another and await the results. We wish to allow the label on a calling program `Pgm` to be different from the label on the program `Pgm′` it calls. Assume `Pgm` is executing for some user U having clearance $\mathcal{L}(U)$.

(a) What restrictions must $\mathcal{L}(U)$, $\mathcal{L}(\texttt{Pgm})$, and $\mathcal{L}(\texttt{Pgm}')$ satisfy to ensure that a user U can learn the contents of only those flies F where $\mathcal{L}(F) \leq \mathcal{L}(U)$ holds. Give evidence that your restrictions are necessary.

(b) What restrictions must $\mathcal{L}(U)$, $\mathcal{L}(\texttt{Pgm})$, and $\mathcal{L}(\texttt{Pgm}')$ satisfy to ensure that a user U can update the contents of only those files F where $\mathcal{L}(F) \leq \mathcal{L}(U)$ holds. Give evidence that your restrictions are necessary.

8.10 One proposal for ensuring that MLFC Computation Restriction always holds is to make conservative choices for classifications assigned to files that provide inputs to G_F. Suppose that $\mathcal{L}(F) = \langle S_F, C_F \rangle$ holds and the inputs to G_F are from files F_1, F_2, ..., F_N. Under this proposal, the classifications $\mathcal{L}(F_i) = \langle S_i, C_i \rangle$ assigned to each input files F_i would satisfy

$$S_F \leq \max(S_1, S_2, ...S_N)$$
$$C_F \subseteq (\cup_{1 \leq i \leq N} C_i)$$

and therefore $\mathcal{L}(F) \leq \mathcal{L}(F_i)$ for $1 \leq i \leq N$. Discuss the utility of the proposed scheme.

8.11 Discuss which of the following would need to be a trusted subject in a system that enforces the restrictions of §8.1.1 for supporting Multi-level File Confidentiality Policy under some suitable label scheme.

(a) A component that reads from secret files that contain votes cast by individuals and writes the majority choice to some different output file that can be read by all.

(b) A component that copies files having differing labels from one directory to another.

(c) A printer daemon that routes file contents to printers in different locations, where the location selected depends on the file's label.

(d) A database system that reads a table containing all student grades and produces a file containing only those grades that a specific student has received on assignments and exams.

8.12 Consider a system that enforces multi-level integrity. Suppose an attacker that can login to that system will be assigned a clearance that has a low trustworthiness (e.g., R or P). The attacker wishes to block *bona fide* users from accessing files.

(a) How would an object low-water mark policy help or hinder the attacker? Explain the attack if you believe one is possible.

(b) How would a subject low-water mark policy help or hinder the attacker? Explain the attack if you believe one is possible.

8.13 Give a DTE specification for the following policies by defining domains, types, and an access matrix for *Auth*. Assume a simplified form of multi-level confidentiality with labels S (secret) and P (public), where $\mathsf{P} \leq \mathsf{S}$. Also assume the usual rules for access to data by users and their applications—"no read-up" and "no write-down".

(a) A logging service with clearance S that reads files with classification S and writes sanitized information to a log file with classification P.

(b) A labeling service that reads files having any clearance and writes that input back, except every paragraph in the output is prefixed by the classification label of the input file.

(c) A printer service that receives files having any clearance, sends them to an appropriate printer-driver based on the file's clearance (so files with classification S are printed in a room that is only open to users with clearance S), and replies with an acknowledgment.

8.14 An implementation of DTE is being contemplated. The designer has decided to implement *Auth* by using capabilities. Various implementations of capabilities are given in §7.3. Discuss the suitability of each.

8.15 Consider a DTE access matrix that assigns `read` and `write` privileges.

(a) Give an example matrix that cannot be simulated by using a set of rules that govern domains performing read and write operations on objects, where these rules depend only on the clearance of a domain, the classifications of the object, and labels that are partially-ordered.

(b) Given a general characterization of matrices that cannot be simulated.

8.16 CW Read Restriction and CW Write Restriction depend on relation $\not\prime$ being static, which is equivalent to assuming that no company's competitors ever changes.

(a) What business events cause this assumption to be violated?

(b) Under what conditions could two companies c and c' become competitors without violating the Chinese Wall for any employee? Justify your answer.

(c) Under what conditions could two companies c and c' cease being competitors without violating the Chinese Wall policy for any employee? Justify your answer.

8.17 Suppose all of the employees of a consultancy are replaced by programs that are autonomously and spontaneously executed. These programs read and write files associated with various client companies but do not generate other outputs. We remain concerned that information in records from one company is not reflected in updates to files associated with competitors. Assume, however, that an executing program can perform a $forget(c)$ operation, which deletes all information about company c from that program's memory.

(a) Exhibit a sequence of program operations that ought to be permitted because $forget(c)$ operations are performed but would not be permitted if $forget(c)$ operations were not available.

(b) Discuss how to weaken CW Read Restriction and CW Write Restriction to accommodate execution of *forget*(c) operations.

8.18 CW Read Restriction and CW Write Restriction together assume that a set (*viz.* $C_R(e)$) is stored per employee, but no metadata is stored for each file. We now explore more permissive restrictions on execution that depend on maintaining a set per file.

(a) Give weaker restrictions for read and write operations assuming that per-employee and per-file sets are being maintained. Explain what information the per-file set contains and how that information might be kept current in an implementation.

(b) Prove that the restrictions proposed in (a) ensure security goal (8.9) holds throughout execution.

(c) Is there a sequence of reads and writes that are permitted by the new restrictions proposed in (a) but are not permitted by CW Write Restriction? If so, give it.

(d) Is there a sequence of reads and writes that are permitted by the new restrictions proposed in (a) but are not permitted by CW Read Restriction? If so, give it.

8.19 CW Write Restriction rules out executions that do not violate security goal (8.9). An obvious question is whether general schemes exist that are less restrictive. At a minimum, we might expect an improved scheme to satisfy the following properties.

- An employee e is not blocked from reading a file unless that access causes security goal (8.9) to become *false*.
- Some employee can write to at least two files, each associated with a different company.
- Future read or write actions an employee undertakes depend only on information the employee read at some point in the past.

Prove that such an improved scheme cannot exist by showing that for any scheme satisfying the above properties:

> An employee e_1 satisfying $c_1 \in R(e_1)$ necessarily can receive information owned by company c_2 from an employee e_2 satisfying $c_2 \in R(e_2)$ where c_1 and c_2 are competitors.

8.20 CW Read Restriction and CW Write Restriction require that a set $C_R(e)$ be maintained at run-time for each employee e. An alternative is to predefine fixed sets $C_R(e)$ and $C_W(e)$ of companies that are initially and permanently assigned to each employee e, authorizing e to read only those files associated with the companies in $C_R(e)$ and to write only those files associated with the companies in $C_W(e)$.

(a) What condition(s) must $C_R(e)$ and $C_W(e)$ satisfy to ensure that employee e does not violate security goal (8.9)?

(b) Is there a condition involving all of the $C_R(e)$ and $C_W(e)$ sets that ensures security goal (8.9) is preserved and that could allow executions ruled out by the test you proposed in (a)? Give the condition and give a scenario that is allowed by this global condition but is not allowed by the condition you proposed in (a).

8.21 For each transformation procedure *tp* in Figure 8.4:

(a) state what is *pre*(*tp*) for implementing the desired sequencing of *purchase*, *orderRcvd*, then *invRcvd*.

(b) explain what code, if any, in *tp* falsifies *pre*(*tp*), and

(c) explain what code, if any, in *tp* makes *pre*(*tp′*) *true*, where *tp′* is the transformation procedure that is next in the sequence being imposed.

8.22 Explain how the set of transformation procedures in Figure 8.4 defend against the following kinds of fraud.

(a) An untrustworthy employee in the front office getting paid by issuing a purchase order to a fake company, delivering no goods, and later submitting an invoice against that purchase order.

(b) An untrustworthy employee on the loading dock accepting a delivery but not invoking *orderRcvd*, thereby allowing the delivered goods to be stolen instead of transferred into inventory.

(c) A dishonest supplier delivering goods that had not been ordered but are nevertheless accepted, thereby creating grounds to be paid.

(d) An untrustworthy supplier or an accomplice in the billing department submitting a second invoice against a given purchase order, thereby creating grounds to be paid double.

8.23 Compare and contrast each of the following with the role construct that RBAC offers.

(a) Groups, often available for assigning privileges to named sets of users in connection with access control lists.

(b) Compartments, part of the DoD labels used in supporting authorization based on need-to-know.

8.24 For each of the following, either give a scenario where the formula does not hold or give a proof that the formula always holds. Recall that $\sqsubset$ signifies role inheritance.

(a) $R \sqsubset R$

(b) $(R \sqsubset R' \wedge R' \sqsubset R'') \;\Rightarrow\; R \sqsubset R''$

(c) $R \sqsubset R' \;\Rightarrow\; \mathit{RolePrivs}(R) \subseteq \mathit{RolePrivs}(R')$

(d) $(\mu(S) = \mu(S') \;\wedge\; \rho(S) \subseteq \rho(S')) \;\Rightarrow\; \rho^*(S) \subseteq \rho^*(S')$

(e) $(\mu(S) = \mu(S') \;\wedge\; \rho(S) \subseteq \rho(S')) \;\Rightarrow\; \mathit{privs}(S) \subseteq \mathit{privs}(S')$

8.25 The following constraint has been suggested for specifying that privilege p is available only to occupants of one role (without stipulating which role).

$$(\forall R, R' \in \mathit{Roles}\colon\; p \notin \mathit{RolePrivs}(R) \vee p \notin \mathit{RolePrivs}(R'))$$

Does it achieve the desired effect? If not, explain the problem and give a constraint that does achieve the desired effect.

8.26 Give RBAC constraints to enforce the following separation of duty policies.

(a) At most one user is occupying role R at any time.

(b) At most n users are occupying role R at any time.

(c) `EK` is the only user that ever occupies role R.

(d) Only one user ever occupies role R.

(e) Roles R and R' are never both occupied at the same time, even by different users.

8.27 Describe the users, roles, permissions, role inheritance, and constraints for each of the following systems.

(a) A course web site that serves notes, problem sets, solution sets, and grades on assignments and exams.

(b) A bank information system that serves information about customer accounts as well as directory and payroll information about the staff.

(c) A hospital information system that serves patient health records, patient billing information, directory and payroll information about the staff, purchase orders and inventory of drugs and other supplies.

8.28 Is it possible to enforce Multi-level Document Confidentiality Policy (page 196) using role-based access control? Explain why not or describe the implementation.

8.29 What events must be detected and possibly blocked by a reference monitor in order to ensure that each of the following constraints is enforced. For each, describe the minimal set of checks that the reference monitor must make.

(a) $(\forall U \in \mathit{Users}\colon\; R \notin \mathit{UserRoles}(U) \vee R' \notin \mathit{UserRoles}(U))$

(b) $(\forall S \in \mathcal{A}\colon\ R \notin \rho(S) \vee R' \notin \rho(S))$

(c) $(\forall S, S' \in \mathcal{A}\colon\ \mu(S) = \mu(S')\ \Rightarrow\ (R \notin \rho(S)\ \vee\ R' \notin \rho(S')))$

(d) $(\forall S, S' \in \mathcal{A}\colon\ \mathit{locate}(\mu(S)) = \mathit{locate}(\mu(S'))\ \Rightarrow\ (R \in \rho(S)\ \vee\ R \in \rho(S')))$

Notes and Reading

Security policies that are institutionally imposed—our *sine qua non* for a mandatory access control policy—have a long history. Double-entry bookkeeping (an integrity policy) was being used in Venice when Pacioli [14] described this practice in 1494. Secrecy, however, has been the bigger concern, both for preserving technological superiority and for creating surprise. During the Bronze age, Greek city states kept secret the recipe for a napalm-like burning mixture ("Greek Fire") their ships in combat would catapult onto an adversary's to wreak havoc. And Sun Tzu's *The Art of War* [36], written in 300 BC, advocates a secrecy policy when it opines that "the formation and procedure used by the military should not be divulged beforehand". Outside of the military, we find medieval guilds imposing secrecy policies that prevent nonmembers from learning skills needed to enter certain occupations. We also find enterprises throughout history maintaining trade secrets either to protect a competitive advantage or to secure a first-mover advantage through surprise.

The term "mandatory access control" is used in the Orange Book [15, §3.1.1.3] to name authorization policies where access to objects by principals is based on labels assigned by authorized principals. These labels must be "a combination of hierarchical classification levels and non-hierarchical categories" [15, §3.1.1.4]. So the scheme codifies the computerization of (paper) document-classification that was then in use by the U.S. government for protecting confidential information. Quist [32, chapter 2], drawing heavily from an unpublished manuscript by Patterson [30], chronicles the precursors and development of U.S. document-classification schemes, which were derived from the British circa 1917.[27]

Britain in fact had all of the elements for a modern document-classification scheme in place by late in the 19th century. Prior to the Crimean War (1853–1856), the British War Office had been marking documents that should be kept confidential, and by 1894 British Army regulations were distinguishing between markings "Secret" and "Confidential" each of which imposed specific rules for handling and disclosure. An early version of "need to know" appears in an 1868 publication of British Army regulations:

> Access to official records is only permitted to those who are entrusted with the duties of the office or department to which they belong...

[27] The current U.S. document-classification scheme is described in Executive Order 13526 [28] signed by President Barack Obama in December 2009. It is the most recent in a series of Executive Orders concerned with U.S. document classification, starting with Executive Order 8381 [33] signed by President Roosevelt in March 1940 [32, chapter 3].

Peacetime classification of information in Britain commenced with an 1866 report on mines and torpedoes—new technologies that, if known by an adversary, might disrupt the Royal Navy's superiority.

Publication of the Orange Book punctuated a process that began once the U.S. Department of Defense (DoD) started to contemplate moving classified information onto time-sharing systems. An early and widely-cited articulation of the issues appears in the 1970 report [38] from a committee chaired by Ware and convened in Fall 1967 under the auspices of the Defense Science Board.[28] Computers back then were expensive and, thus, had to be shared. So for storing classified documents, DoD required

- a system that could support concurrently logged-in users having different clearances and accessing objects that had different classifications, and
- an assurance argument to establish that the system's access controls could not be circumvented either by *bona fide* users or by outsiders.

The Adept-50 time-sharing system [39] was an early and notable attempt to address these DoD needs. That system, which was operational by 1969, was developed at System Development Corporation (SDC) under an ARPA[29] contract. The authorization policy Adept-50 enforced was based on a *high-water mark* that the system maintained for each process. This high-water mark was initialized to a value below both (i) the clearance held by the user that started the process and (ii) the label assigned to the terminal that was attached to the process. A `read` would be permitted by a process if its high-water mark was greater than the label on the file being read. A `write` by a process would set label $\mathcal{L}(F)$ on the file F being written to the current value of the high-water mark for the process. Adept-50 also supported a `change` command for reducing the security label associated with an object. By executing `change`, a program running in Adept-50 could exfiltrate classified information—read it from one file, write it to another file (causing the file's label to be set appropriately), and then invoke `change` to decrease the label on that output file so the classified information now could be read by any process.

Frustrated by a lack of progress in creating secure time-sharing systems, USAF Major Schell commissioned a study, chaired by Glaser (at Case Western Reserve University), to propose a research and development plan. That committee's final report [1], published in 1972, is today known as the Anderson Report, named after the committee member who managed the study and did a lot of the writing. The report advocates (among other things) that a small security kernel be the only software involved in enforcing a system's authorization policies—an architecture that Schell had been advocating to replace the prevailing view that

[28] Much has been written about this early history. Mackenzie and Pottinger [25] is an excellent reference.

[29] The Advanced Research Projects Agency (ARPA) was created in 1958 to fund research in support of DoD and, thereby, help to avoid technological surprise. ARPA is the predecessor of today's Defense Advanced Research Projects Agency (DARPA), which remains a significant source of funds for U.S. computer science research.

extant time-sharing systems could simply be augmented with further checks. Only with a small security kernel would thorough analysis and testing be possible, thereby providing an assurance argument for the enforcement mechanism. An assurance argument for the entire time-sharing system then would be obtained by combining the assurance for the enforcement mechanism with a proof that the authorization policy[30] implies the intended security properties. The Anderson Report also popularized the term Trojan horse, which its appendix I "Security threats and penetration techniques" attributes to Dan Edwards, who has NSA's representative to committee.

Two research groups were subsequently funded to develop formal models, define security policies, and prove these polices ensured that information labeled as classified could not be read by users lacking suitable clearances. Bell and La Padula, working at MITRE, published their proposal in 1973 [6, 5]; it became the basis for virtually all DoD computer security work for the next decade. Walter et al., working at Case Western University, independently developed essentially the same restrictions (couched in terms of repositories and agents rather than files and processes) and published their work [37] a few months later. The rules we give in §8.1.1 for enforcing multi-level confidentiality are based on Feiertag et al. [16], which gives a succinct reformulation of Bell and La Padula [6, 5]. Our MLFC Computation Restriction and the analysis that supports it are new, although Landweher's 1981 survey [22] notes that aggregations could warrant a higher classifications than their components. Landwehr [22] is also worth studying to position Bell and La Padula's work relative to other (early) formal models for computer security.

Trusted subjects were proposed in Bell and La Padula [7], which extends [6, 5] for a secure Multics [29]. (Walter et al. [37] is credited with developing the access policy given there for Multics tree-structured directories. These files have a specific semantics and thus warranted special treatment.) Bell and La Padula [7, section IV] also identified some limitations with the definition of security in [6, 5], which ignored covert timing channels, information corruption, and denial of service.

An attempt by Landwehr, Heitmeyer, and McLean [23] at the Naval Research Laboratory to establish security of a military-message system exposed other problems with the Bell and La Padula [7] model. They found that extensive use of trusted subjects was required for this application. But that meant the "Basic Security Theorem" proved in Bell and La Padula [7] said little about whether the intended security policy was actually being enforced—message security depended on what the trusted subjects did, yet trusted subjects could do anything in the Bell and La Padula [7] model. From this experience, Landwehr et al. [23] concludes that security policies must be formulated for specific applications rather than being formulated in terms of a pre-determined set of application-independent abstractions supported by a general-purpose secure computing system.

[30] The authorization policy here is presumed to be specified in terms of some formal model of the security kernel.

McLean's further criticisms [26] of the "Basic Security Theorem" and his infamous system Z, which automatically declassified objects to make all read and write access requests appear legal [27], led to heated debate [4]. The same "Basic Security Theorem" proved in Bell and La Padula [7] for Multics could be proved for system Z. That observation raised questions about assurance that the "Basic Security Theorem" actually provided, and it called attention to the necessity of a Tranquility Assumption.

Biba [8] proposed the rules in §8.1.2 for integrity of labeled data, remarking that these rules are the "compliment" or "dual" of the Bell and La Padula rules [6, 5] for protecting confidentiality of labeled data: MLFI Write Restriction and MLFI Read Restriction are just MLFC Write Restriction and MLFC Read Restriction with "read" and "write" interchanged. Although Biba [8] does not explore the mathematics of this connection between integrity and confidentiality, it does have a mathematical basis. The confidentiality labels used by Bell and La Padula [6, 5] form a lattice, and this lattice is the dual of the lattice of integrity labels employed by Biba [8], because relation $\preceq_I$ for integrity is the inverse of relation $\preceq$ for confidentiality.

Use of an access matrix for specifying MAC policies was originally proposed by Boebert and Kain [9, 10] for implementing authorization in Sidewinder, a system purportedly being developed to execute Ada[31] programs securely.[32] The scheme, called *type enforcement*, was a radical (and politically perilous, given their source of research funding) break with the Orange Book's dogma, although it was entirely consistent with lessons reported by Landwehr, Heitmeyer, and McLean [23] from the military-message system. No longer would trusted subjects be needed to sidestep the restrictions on read-up and write-down, which meant type enforcement supported assurance by mechanism rather than by fiat. Moreover, (i) type enforcement provided means to arrest the tendency for labels on data to drift ever higher, (ii) the Principle of Least Privilege could be enforced for programs that required the ability to read-up or write-down, and (iii) information could be forced to traverse prescribed pipelines of programs. Domain and type enforcement [2] was obtained by augmenting type enforcement with a level of abstraction to facilitate a UNIX implementation that would be backward compatible and would be easy to administer; SELinux [31] today continues to support the approach.

As noted earlier, the Orange Book was developed under the auspices of DoD. Lipner [24] in 1982 gives the first indications that its approach to authorization might not be a panacea, because commercial institutions have different computer security needs than military institutions. But a 1987 paper [12] by Clark (a

[31] Ada was a programming language developed under the auspices of DoD. Standardizing on a well designed language was expected to reduce costs and increase reliability of embedded and real-time software.

[32] Sidewinder actually was intended to defend against supply-chain attacks hosted in a personal computer that would be executing high-grade cryptographic routines and serve as a frontend to a Multics system. With such a frontend, the Multics system no longer had to be trusted to ensure that messages it sent could not be intercepted and read by attackers. McAfee Firewall Enterprise is Secure Computing's Sidewinder product. (Secure Computing was acquired by McAfee in 2008.)

computer scientist) and Wilson (an accountant) is where that discrepancy is first forcefully argued. Completely different forms of malfeasance (e.g., fraud and error, rather than disclosure) are the primary concern outside of the military; the Orange Book's partially ordered labels are not a useful basis for access control, here.

Clark and Wilson [12] puts the research community on notice—multi-level security is not the only kind of MAC but just one kind, and a new (or significantly revised) Orange Book would be needed to support commerce. Further corroboration for this thesis comes with the publication of Brewer and Nash [11], which introduces computer security researchers to yet another useful class of MAC policies for commercial institutions. These Chinese Wall policies are not only orthogonal to the multi-level security described in the Orange book but they differ from the commercial policies of Clark and Wilson [12].

Chinese Wall[33] policies were originated by U.S. investment-industry regulators following the 1929 stock market crash that began the Great Depression. A so-called Chinese Wall gave the public assurance that a brokerage was being prevented from profiting at the expense of its customers. Separation of duty, in fact, had already enjoyed a long history in governance, where it is called "separation of powers". The U.S. Constitution (drafted in 1789), for example, stipulates a tripartite structure comprising a legislative branch (to make laws), an executive branch (to enforce laws), and a judiciary branch (to interpret laws). Tripartite governing structures, which is attributed to the Age of Enlightenment political philosopher Baron de Montesquieu [13], had earlier been adopted by the Dutch and English. Still further back, we see governments of the Roman Republic and Greek city-states in antiquity employing separation of powers. Among the earliest discussions of separation of duty in connection with computer security is the "separation of privilege" principle discussed in Saltzer and Schroeder [34], which credits a 1973 conversation with Roger Needham.

The other defenses Clark and Wilson [12] suggests—audit and the use of well-formed transactions—derive from classic accounting controls, which date back to the beginnings of commerce and taxation. The Egyptians and Babylonians employed audit schemes to keep track of warehouse contents so they could reconcile inventory with deposits and withdrawals. And by the 1400's, various methods were in use for keeping records in banking houses. Double-entry bookkeeping was among these; it implements an append-only log by stipulating (i) that entries be in ink, hence permanent, and (ii) that erroneous entries be corrected not by removing or changing them but by making further compensating entries. Double-entry bookkeeping also implements a general framework for defining well-formed transactions in support of virtually all of today's business activities.

Early operating systems did not support roles per se. However, early systems did support groups, and groups (like individuals) were principals that could be assigned privileges. Some systems only supported a predefined set of groups;

[33] The term is attributed to U.S. President Franklin D. Roosevelt who, shortly after he was elected in 1933, used the phrase "Chinese wall of silence" to describe the isolation being sought for different principals within a single institution. [21, page 81].

other systems allowed groups to be created and populated as needed, either by system administrators or by users. By the mid 1980's, Landwehr et al. [23], for their military-message system, proposed a security model that foreshadows modern role-based access control:

> Role—the job a user is performing, such as downgrader, releaser, distributor, and so on. A user is always associated with at least one role at any instant, and the user can change roles during a session. To act in a given role, the user must be authorized for it. Some roles may be assumed by only one user at a time (e.g., distributor). With each role comes the ability to perform certain operations.

The conclusion of Landwehr et al. [23], however, was only that different applications required different authorization policies—it did not opine about whether roles, sessions, and constraints served well as a general-purpose model for authorization.

Ferraiolo and Kuhn were the first to argue that role-based access control would be broadly applicable. Concerns being voiced (e.g., [23, 9, 10, 12]) about the applicability of the security model prescribed by the Orange Book, had let Ferraiolo et al. to survey[34] [18] the security needs of civilian government agencies and commercial enterprises. The responses indicated that these non-military institutions required means to associate privileges with roles (rather than with individuals) and that such functionality was not easily implemented using existing systems. A form of role-based access control thus seemed like it would better serve those surveyed than the Orange book's label-based rules or than Clark and Wilson's transformation procedures.

An initial proposal by Ferraiolo and Kuhn [19] focused on roles and inheritance [19]; it generalized *named protection domains*, which had been described in Baldwin [3] as a basis for authorization in ANSI SQL databases. Constraints were added later [17], once roles proved inadequate for formalizing certain separation of duty policies. Our treatment in §8.4 is derived from Sandhu et al. [35], an early and influential effort to structure role-based access control in terms of simpler models: $RBAC_0$ supports roles, sessions, and privileges; $RBAC_1$ adds role inheritance to $RBAC_0$; $RBAC_2$ adds constraints to $RBAC_0$; and $RBAC_3$ combines $RBAC_1$ and $RBAC_2$. ANSI standard (INCITS 359-2004, approved Feb 2004) evolved from $RBAC_3$ (see [20]). The ANSI standard is supported by operating systems (e.g., SELinux, Microsoft's active directory, FreeBSD) and by database management systems (Microsoft SQL server, Oracle DBMS, SAP R/3).

Bibliography

[1] James P. Anderson. Computer security technology planning study. Technical Report ESD-TR-73-51, Electronic Systems Division (AFSC), Hanscom

[34] The study covered 28 organizations, including 17 federal agencies, 10 corporations, and 1 state agency.

Field, Bedford, MA, October 1972.

[2] Lee Badger, Daniel F. Sterne, David L. Sherman, Kenneth M. Walker, and Sheila A. Haghighat. Practical domain and type enforcement for UNIX. In *Proceedings of 1995 IEEE Symposium on Security and Privacy*, pages 66–77. IEEE Computer Society Press, 1995.

[3] Robert W. Baldwin. Naming and grouping privileges to simplify security management in large databases. In *Proceedings of 1990 IEEE Symposium on Security and Privacy*, pages 116–132. IEEE Computer Society Press, May 1990.

[4] D. Elliott Bell. Concerning modeling of computer security. In *Proceedings of 1988 IEEE Symposium on Security and Privacy*, pages 8–13. IEEE Computer Society Press, 1988.

[5] D. Elliott Bell and Leonard J. La Padula. Secure computer systems: A mathematical model. Technical Report ESD-TR-73-278, Volume II, Electronic Systems Division (AFSC), Hanscom Field, Bedford, MA, November 1973.

[6] D. Elliott Bell and Leonard J. La Padula. Secure computer systems: Mathematical foundations. Technical Report ESD-TR-73-278, Volume I, Electronic Systems Division (AFSC), Hanscom Field, Bedford, MA, November 1973.

[7] D. Elliott Bell and Leonard J. La Padula. Secure computer systems: Unified exposition and MULTICS interpretation. Technical Report EDS-TR-75-306, Electronic Systems Division (AFSC), March 1976.

[8] K. J. Biba. Integrity consideration for secure computer systems. Technical Report MTR-3153, MITRE Corporation, Bedford, MA, June 1975.

[9] W. E. Boebert and R. Y. Kain. A practical alternative to hierarchical integrity policies. In *Proceedings of the 8th National Computer Security Conference*, pages 18–27. U.S. Government Printing Office, October 1985.

[10] William E. Boebert and Richard Y. Kain. A further note on the confinement problem. In *1996 International Carnahan Conference on Security Technology*, pages 198–202, 1996.

[11] David F. C. Brewer and Michael J. Nash. The Chinese Wall security policy. In *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, pages 206–214. IEEE Computer Society Press, May 1989.

[12] David D. Clark and David R. Wilson. A comparison of commercial and military computer security policies. In *Proceedings of 1987 IEEE Symposium on Security and Privacy*, pages 184–194. IEEE Computer Society Press, 1987.

[13] Baron de Montesquieu. De l'esprit des lois. Republished as *Montesquieu: The Spirit of Laws*, Cambridge Texts in the History of Political Thought, Cambridge University Press, 1989, 1748.

[14] Luca Bartolomeo de Pacioli. *Summa de arithmetica, geometria, proportioni et proportionalità*. Venice, 1494.

[15] Department of Defense. *Department of Defense Trusted Computer System Evaluation Criteria*. DoD 5200.28-STD, Supercedes CSC-STD-001-83 dated 15 August 1984, Library Number S225,711.

[16] R. J. Feiertag, K. N. Levitt, and L. Robinson. Proving multilevel security of a system design. In *Proceedings of the Sixth ACM Symposium on Operating Systems Principles*, SOSP '77, pages 57–65, New York, NY, USA, 1977. ACM.

[17] David F. Ferraiolo, Janet A. Cugini, and D. Richard Kuhn. Role-based access control (RBAC): Features and motivations. In *Proceedings of 11th Annual Computer Security Applications Conference*, pages 241–248. IEEE Computer Society Press, December 1995.

[18] David F. Ferraiolo, Dennis M. Gilbert, and Nickilyn Lynch. Assessing federal and commercial information security needs. Technical Report NISTIR 4976, National Institute of Standards and Technology, Computer Systems Laboratory, Gaithersburg, Maryland, November 1992.

[19] David F. Ferraiolo and D. Richard Kuhn. Role-based access controls. In *Proceedings of 15th National Computer Security Conference*, pages 554–593. National Institute of Standards and Technology, National Computer Security Center, October 1992.

[20] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information System Security*, 4(3):224–274, August 2001.

[21] Anthony Hilton. *City within a State: A Portrait of Britain's Financial World*. I. B. Tauris & Company Limited, 1987.

[22] Carl E. Landwehr. Formal models for computer security. *ACM Computing Surveys*, 13(3):247–278, September 1981.

[23] Carl E. Landwehr, Constance L. Heitmeyer, and John D. Mclean. A security model for military message systems. *ACM Transactions on Computer Systems*, 2(3):198–222, August 1984.

[24] Steven B. Lipner. Non-discretionary controls for commercial applications. In *Proceedings of 1982 IEEE Symposium on Security and Privacy*, pages 2–10. IEEE Computer Society Press, 1982.

[25] Donald Mackenzie and Garrel Pottinger. Mathematics, technology, and trust: Formal verification, computer security, and the U.S. military. *IEEE Annals of the History of Computing*, 19(3):41–59, July 1997.

[26] John McLean. A comment on the 'Basic Security Theorem' of Bell and La Padula. *Information Processing Letters*, 20(2):67–70, February 1985.

[27] John McLean. Reasoning about security models. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 123–133. IEEE Computer Society Press, May 1987.

[28] Barack Obama. Classfied national security information. Executive Order EO 13526, The White House, December 2009. https://www.fas.org/irp/offdocs/eo/eo-13526.htm.

[29] Elliott I. Organick. *The Multics System: An Examination of its Structure*. MIT Press, 1972.

[30] Andrew Patterson, Jr. "CONFIDENTIAL" – The beginning of defense-information marking. Unpublished manuscript. Sterling Chemistry Laboratory, Yale University, April 1980.

[31] SELinx Project. http://selinuxproject/page/Main_Page.

[32] Arvin S. Quist. Security Classification of Information, Volume 1. Introduction, History, and Adverse Impacts. Technical Report ORCA–12, Oak Ridge Classification Associates, LLC, September 2002. http://www.fas.org/sgp/library/quist/index.html.

[33] Franklin D. Roosevelt. Defining certain vital military and naval installations and equipment. Executive Order EO 8381, The White House, March 1940. https://www.fas.org/irp/offdocs/eo/eo-8381.htm.

[34] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, March 1975.

[35] Ravis S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E Youman. Role-based access control models. *Computer*, 29(2):38–47, Feb 1996.

[36] Sun Tzu. *The Art of War*. Courier Dover Publications, 2013.

[37] K. G. Walter, W. F. Ogden, W. C. Rounds, F. T. Bradshaw, S. R. Ames, and D. G. Shumway. Primitive models for computer security. Interim Technical Report ESD-TR-4-117, Case Western Reserve University, 1974. NTIS AD-778 467.

[38] Willis H. Ware. Security control for computer systems: Defense Science Board Task Force on Computer Security. Technical Report R-609-1, Rand Corporation, Santa Monica, CA, February 1970.

[39] C. Weissman. Security controls in the ADEPT-50 time-sharing system. In *Proceedings of the 1969 Fall Joint Computer Conference*, AFIPS Conference Proceedings, pages 119–133. AFIPS Press, 1969.